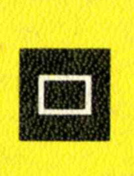
programación del MICROPROCESADOR



ELIZABETH A. NICHOLS
JOSEPH C. NICHOLS
PETER R. RONY

BOIXAREU EDITORES

marcombo
BOIXAREU EDITORES

PROGRAMACION DEL MICROPROCESADOR Z-80

cemvsa
Pj Antonio Gassol 18 Barcelona-5

Elizabeth A. Nichols
Joseph C. Nichols
Peter R. Rony

PROGRAMACION DEL MICROPROCESADOR Z-80

Traducido por:
Manuel Puigbó Rocafort
Perito Industrial

marcombo
BOIXAREU EDITORES

BARCELONA - MEXICO

Título de la obra original:

Z-80 Microprocessor
Book 1. Programming

por Elizabeth A. Nichols, Joseph C. Nichols, and Peter R. Rony

Gran Via de les Corts Catalanes, 594
BARCELONA-7

ISBN: 84-267-0408-5

ISBN: 0-672-21609-4, Howard W. Sams & Co., Inc, edición original

Dep. Legal: B. 36844-80

Impreso en España
Printed in Spain

T. G. Portavella, S. A.
Diputación, 427 - Barcelona-13

Prefacio

La revolución de la microelectrónica está aquí, y va en aumento. Todo esto empezó con el desarrollo del transistor, un pequeño amplificador de baja potencia, que reemplazó los tubos de vacío, que tenían un alto consumo, y fueron utilizados en la primera generación de computadores. Los transistores se han convertido en los bloques básicos para fabricar los circuitos de los computadores, debido a la facilidad de aplicar estos en los circuitos de lógica digital. Los transistores se combinan para formar puertas; las puertas se combinan para formar básculas, contadores, sumadores y otras funciones lógicas; y estos a su vez, se combinan para formar la memoria, el control y las unidades aritmética y lógica que constituyen la unidad central de proceso de un computador (CPU). Así, el número de transistores en un circuito lógico se ha convertido en una medida razonable de su complejidad funcional.

En 1959 se desarrollaron los primeros circuitos integrados constituidos de pequeños grupos de transistores planar y se fabricaron en finas obleas de silicio o germanio. Esto empezó la era de la Integración en Pequeña Escala (SSI) (Small Scale Integration), en la cual se podían incorporar en un solo circuito integrado 12 ó menos puertas. Desde 1959, el número de transistores en los circuitos integrados avanzados se ha doblado, por lo menos, cada año. Hoy en día se puede disponer de circuitos que contienen 262, 144 elementos y la tecnología está lejos de sus límites teóricos. La CPU Z-80 y sus circuitos de soporte, introducidos por Zilog en 1976, representan lo mejor de los procesadores de 8 bits. Zilog está actualmente desarrollando un sucesor de la línea del Z-80, la serie Z-8000 con su

CPU y circuitos de soporte. Sin embargo, el Z-8000 será una CPU de 16 bits con una capacidad de cálculo comparable a los minicomputadores de la gama media, lo que constituye un salto significativo en su capacidad. Y esto es sólo el principio. La revolución real se manifestará en la proliferación exponencial de productos y servicios que dependen de la microelectrónica.

Este libro es el primero de dos volúmenes que tratan de la programación e interface del microprocesador Z-80. El Libro 1* trata del software: programación en lenguaje ensamblador y en lenguaje máquina. El Libro 2 cubre el interface de los circuitos digitales con la CPU del Z-80, PIO y circuitos CTC. Estos libros están orientados al trabajo de laboratorio y están diseñados para dar un método de programación del microcomputador e interface del mismo. El mayor énfasis se ha puesto en el aprendizaje mediante la experimentación. Cada nuevo tópico introducido se refuerza con un trabajo de laboratorio que muestra no sólo cuando las ideas tienen éxito, sino también cuando fallan, y donde están estos fallos.

El Libro 1 no necesita conocimientos en la ciencia del computador. El Libro 2, sin embargo, supone una cierta familiaridad con los tópicos que se cubren en el libro 1. En ambos libros, los tópicos se presentan en el orden en que los autores consideran más adecuado para el auto-aprendizaje.

Se proporcionan las respuestas de todos los ejercicios, y se hace todo lo posible para poner preguntas anticipadas y extensiones lógicas a los experimentos.

Para mejorar la orientación de laboratorio en los libros, los experimentos utilizan un microcomputador Z-80 sofisticado manufacturado por SGS-ATES, llamado el Nanocomputador. El Nanocomputador es un computador educacional excelente, debido a que es fácil de manejar por un novato, pero incorpora un suficiente número de opciones, flexibilidad, expandibilidad y sofisticación para mantener el interés del utilizador más experimentado.

Los autores agradecen a los muchos miembros del staff de SGS-ATES en Milán, Italia: R. Baldoni, A. Cattania, B. Facchi, F. Luraschi, C. Wallace, y especialmente A. Watts cuyas muchas ideas y experiencia técnica en el Nanocomputador mejoraron enormemente estos libros. También deseamos dar las gracias a C. Edson y U. Broggi de SGS-ATES en USA que hicieron progresar este proyecto, haciendo de unión entre los esfuerzos estadounidenses e italianos en este proyecto.

ELIZABETH A. NICHOLS
JOSEPH C. NICHOLS
PETER R. RONY

**Nota del Editor*: Los comentarios que puedan surgir en la obra de "Libro 1" y "Libro 2" se han respetado en la versión castellana de la presente edición. No obstante, debemos indicar que actualmente sólo existe traducción al castellano del "Libro 1".

Indice

CAPITULO 5

CAPITULO 6

CAPITULO 7

CAPITULO 8

CAPITULO 9

CAPITULO 10

CAPITULO 11

APENDICE A

APENDICE B

APENDICE C

APENDICE D

APENDICE E

APENDICE F

APENDICE G

1

Códigos digitales

INTRODUCCION

Antes de que Vd. empiece a programar su microcomputador, es necesario que entienda como se convierten los números binarios de 8 bits en código hexadecimal, y viceversa, así como que conozca algunos hechos básicos acerca de los códigos digitales.

OBJETIVOS

Al completar este capítulo, Vd. será capaz de hacer lo siguiente:

- Discutir qué significa el término *comunicación.*
- Definir *bit.*
- Definir *código binario.*
- Definir *código digital.*
- Definir el *código hexadecimal.*
- Convertir un número binario de 8 bits en un número hexadecimal de dos dígitos.
- Convertir un número hexadecimal de dos dígitos a un número binario.
- Distinguir entre los sistemas de contaje binario, hexadecimal y decimal.
- Dar varios códigos digitales distintos.
- Indicar varios dispositivos distintos de dos estados.

- Proporcionar un ejemplo en el cual la cantidad, bits por segundo, es una medida del flujo de la información.

LENGUAJES, COMUNICACIONES E INFORMACION

Una de las características más importantes que posee cualquier organismo biológico (animales de mayor grado) es la habilidad de comunicarse con otros organismos de la misma especie. La habilidad de comunicarse, la cual da a muchos organismos animales una ventaja definitiva de supervivencia –en el sentido Darwiniano del término– se encuentra en muchas criaturas multicelulares, empezando con los insectos y progresando hasta el hombre. Con los insectos, incluyendo la danza de la abeja y formas de comunicación química mediante notables agentes químicos llamados *pheromones.* El hombre se puede comunicar con la ayuda de sus cinco sentidos, como demuestran los individuos en inferioridad que han perdido uno o más de sus sentidos, pero que a pesar de todo son altamente comunicativos con los que les quedan.

Suponiendo que un individuo desea comunicarse con otro mediante el sentido del oído y la utilización de la palabra, está claro de que debe existir algún acuerdo general que determine como se debe de interpretar una palabra hablada, por otro individuo que la oye. A través de centenares de años, las diferentes regiones en el mundo han desarrollado su propio consenso con respecto al significado de sonidos específicos y su transcripción en el papel. Llamamos a este consenso un *lenguaje,* tal vez, un *lenguaje extranjero.* Existen miles de lenguajes diferentes, aunque solamente un número relativamente modesto son de uso generalizado. La popularidad de un lenguaje determinado puede aumentar y disminuir a través del transcurso de varios centenares de años. El latín, que fue el lenguaje dominante en Europa, es considerado actualmente como un lenguaje muerto, sin embargo ha influenciado en muchos de los lenguajes europeos en muy profundas formas.

La *comunicación* puede definirse como impartir la enseñanza, transmisión o intercambio de ideas, conocimiento, información, etc. (mediante la palabra, la escritura o signos).[1]* Es una de las más importantes y características actividades del hombre. Como ha explicado James Martin en su excelente libro, *Telecommunications and the Computer,*[3] la capacidad de las uniones de telecomunicación, medidas mediante una cantidad llamada bits por segundo, ha avanzado en paralelo con la civilización a través de los últimos cien años. La capacidad de tales enlaces ha cambiado desde una relación de 1 bit/segundo en 1840 a 50.000.000 de bits/segundo en 1970, es decir que ha doblado cada 5,08 años. Martin también ha

*Véase Apéndice G para todas las referencias.

señalado que la suma total de los conocimientos humanos cambió muy lentamente antes del relativamente reciente principio del pensamiento científico. En el 1800, se ha estimado que la suma total fue doblando cada 50 años; en el año 1950, doblando cada 10 años; y que en 1970 doblara cada 5 años.

Un *lenguaje*, el cual puede definirse como el conjunto de palabras y de métodos de combinación de las palabras utilizadas por una nación, pueblo o raza,[1] es solamente una forma de comunicación. Los geroglíficos egipcios, puntuaciones coreográficas, ecuaciones y símbolos matemáticos, las señales de humo de los indios americanos, el lenguaje de signos empleado por los mudos y el código de Morse, son otras formas de comunicación utilizadas por el hombre.

CODIFICACION BINARIA

La "explosión de la información" habría inundado al hombre, por lo menos en los países más avanzados, si no hubiera sido por el uso de la *Codificación de Dos Estados* para representar todas las clases de información, tal como los diez números decimales (0 al 9), las veintiséis letras del alfabeto inglés (de la A a la Z), operaciones, símbolos, movimientos, etc. Llamamos a esta codificación de dos estados, *codificación binaria.*

La codificación binaria puede ser representada o manifestarse mediante cualquier tipo de dispositivo de dos estados, tal como una lámpara encendida o apagada, un interruptor abierto o cerrado, una tarjeta de computador perforada o no, un núcleo magnético magnetizado como "norte" o "sur", o una región de una cinta o disco magnéticos; dos niveles distintos de tensión, dos niveles distintos de corriente, dos frecuencias distintas; las palabras SI y NO; o los símbolos abstractos 0 (apagado) y 1 (encendido). La importancia de la codificación binaria reside en el hecho de que es posible construir dispositivos que pueden cambiar de estado muy rápidamente, en tiempos tan cortos como 5 nanosegundos (0,000000005 segundo). Este dispositivo puede en principio, manipular, transmitir o recibir información a la velocidad de 200 millones de bits por segundo. Treinta y dos de estos dispositivos, funcionando simultáneamente, pueden manipular seis mil cuatrocientos millones de bits por segundo. Esta es la capacidad básica que ha permitido a la sociedad el guardar, manipular y comunicar cantidades enormes de información.

BIT

La unidad elemental de información es llamada *bit,* que es la abreviación de la palabra **BI**nary digi**T** (dígito binario). Se puede pensar que un bit es una lámpara que puede estar encendida o apagada en un determinado tiempo. Así, un bit se

puede representar como una lámpara que está encendida o apagada. En lugar de dibujar lámparas, podemos representar cuando una lámpara está encendida mediante el símbolo 1 y una lámpara apagada mediante el símbolo 0.

Así, un bit es igual a una decisión binaria o la designación de uno de los dos posibles valores o estados (tales como 0 ó 1).

La información se representa típicamente mediante series de bits. Así,

1 0 0 0

representa el número 8 decimal en código binario. La serie de bits,

1 1 0 0 0 0 0 1

representa la letra A en código ASCII de 8 bits. Discutiremos estos dos códigos brevemente.

CODIGOS DIGITALES

Un *código digital* está definido como un sistema de símbolos que representan los valores de los datos y que está constituido por un lenguaje especial que un computador o un circuito digital puede entender y utilizar.[3] Los códigos digitales pueden ser considerados como los "lenguajes" digitales que permiten almacenar, manipular y comunicar la información. Tal y como existen numerosos lenguajes hablados, también existe una variedad de códigos digitales. Estos códigos pueden ser subdivididos en varias categorías importantes:

Categoría 1. Códigos utilizados por los circuitos electrónicos para realizar varias operaciones digitales. Ejemplo: código binario.

Categoría 2. Códigos utilizados para convertir los números decimales del 0 al 9 a forma digital. Ejemplos: código binario, binario codificado en decimal (BCD) y código de Gray.

Categoría 3. Códigos utilizados para convertir números decimales, las 26 letras del alfabeto inglés, símbolos y operaciones a forma digital. Ejemplos: código ASCII, código EBCDIC y código Baudot.

Categoría 4. Códigos de instrucción utilizados por los grandes computadores, minicomputadores y microcomputadores que hacen que los computadores realicen una determinada secuencia de operaciones. Ejemplos: código de instrucción del IBM 370, código de instrucción del PDP 8/E, código de instrucción del Z-80.

En esta serie de módulos pondremos una atención particular a cuatro códigos: el código binario, binario codificado en decimal (BCD), código ASCII y el código de las instrucciones para el chip del microprocesador Z-80.

CODIGO BINARIO

El código digital más simple es el de dos estados, o binario, código que está formado de dos estados 0 y 1. Llamamos a estos dos estados, estado *lógico 0* y estado *lógico 1*. En el código binario el cero decimal se representa por el 0 lógico y el número decimal 1 mediante el número lógico 1. Esto debe ser muy claro. ¿Cómo se representan los números decimales como 3, 17, 568, etc., utilizando el código binario? La respuesta es que nosotros utilizamos una serie de bits para construir un *sistema binario de contar* que está formado de una base de dos. Por ejemplo, el número binario 11101_2, en el cual el subíndice (2) representa el sistema de contaje binario, es equivalente a

11101 (2) = (1×2**4) + (1×2**3) + (1×2**2) + (0×2**1) + (1×2**0) = 29 (10)

en donde usted debe recordar que A**B es equivalente a A^B. Así,

2**4 = 16 en notación decimal = 16_{10}
2**3 = 8 en notación decimal = 8_{10}
2**2 = 4 en notación decimal = 4_{10}
2**1 = 2 en notación decimal = 2_{10}
2**0 = 1 en notación decimal = 1_{10}

Así,

11101 (2) = 16 (10) + 8 (10) + 4 (10) + 0 + 1 (10) = 29 (10)

en donde el subíndice (10) asociado con estos números representa el sistema de contaje decimal, un sistema que está formado de una base de 10. A continuación está una pequeña tabla que le muestra como convertir números decimales a números binarios.

Número decimal	Número binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000

Así, una serie de cuatro dígitos binarios, o bits, puede representar cualquiera de los dieciséis números decimales del 0 al 15. Los números decimales mayores de quince necesitan bits adicionales, como se muestra en la siguiente tabla:

Número decimal	Número binario
0	0
1	1
2	10
3	11
4	100
7	111
8	1000
15	1111
16	10000
31	11111
32	100000
63	111111
64	1000000
127	1111111
128	10000000
255	11111111
256	100000000
511	111111111
512	1000000000
1023	1111111111
1024	10000000000
2047	11111111111
2048	100000000000
4095	111111111111
4096	1000000000000
8191	1111111111111
8192	10000000000000
16,383	11111111111111
16,384	100000000000000
32,767	111111111111111
32,768	1000000000000000
65,535	1111111111111111

Así, un número binario de 8 bits puede codificar 256 números decimales diferentes, que van desde 0 a 255_{10}, o 256 "cosas" diferentes, no importa lo que puedan ser (instrucciones, dispositivos, impulsos, etc.)

El Z-80 es un chip microprocesador de 8 bits que tiene 16 bits de dirección de memoria, y una palabra de direccionamiento de 8 bits para los dispositivos de I/O (Entrada/Salida).

Esto significa que puede direccionar directamente 65.536 posiciones distintas de memoria y que puede generar por lo menos 256 impulsos de I/O o direcciones de dispositivos.

CODIGO HEXADECIMAL (HEX)

Puede ser difícil el recordar números binarios que contienen muchos bits. Por ejemplo, ¿puede usted recordar el siguiente número binario de 8 bits,

1 0 0 1 1 1 0 1

después de haberlo mirado solamente durante un segundo? Rápido, ¡tápelo o mire a otro lado! Considere también el problema de recordar una lista de los siguientes números de 8 bits:

1 1 0 1 1 0 1 0
1 1 1 0 0 1 0 1
0 1 1 0 1 0 0 1
1 0 1 0 1 0 1 1

Probablemente usted sacará la conclusión de que debe de existir un mejor modo de recordar los números binarios de 8 bits. Estamos utilizando aquí números binarios de 8 bits debido a que usted los encontrará frecuentemente cuando empiece a programar el microcomputador Z-80 de 8 bits.

Una forma de recordar números binarios de varios bits consiste en la utilización del *código hexadecimal*. El término *hex* es simplemente una abreviación de la palabra *hexadecimal*. Los códigos hexadecimales se refieren al *sistema de contaje hexadecimal*, un sistema que está formado de una base de 16. El sistema de contaje hexadecimal consta de dieciséis símbolos diferentes: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. Tal y como hicimos con los números decimales, es posible convertir números hexadecimales a números binarios:

Número decimal	Número hex	Número binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	0001 0000
17	11	0001 0001

18	12	0001 0010
19	13	0001 0011
20	14	0001 0100
21	15	0001 0101
22	16	0001 0110
23	17	0001 0111
24	18	0001 1000
32	20	0010 0000
40	28	0010 1000
48	30	0011 0000
56	38	0011 1000
63	3F	0011 1111

Hemos agrupado los números binarios de 8 bits en grupos de cuatro bits cada uno para ayudarle a comprender como se realiza la conversión de un número hexadecimal a binario. El espacio entre cada grupo de cuatro bits no afecta al valor del número, pero hace que el número binario se pueda leer más fácilmente y se ha convertido en una norma.

Ahora nos preguntamos cómo se puede convertir un número binario de 8 bits en el código hex. El procedimiento para realizar esto necesita tres pasos:

1. Escribir el número binario de 8 bits completo.
2. Separar este número binario de 8 bits en dos grupos con cuatro dígitos binarios en cada grupo.
3. Substituir el dígito hex equivalente

   0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

   para cada grupo de cuatro bits.

Habiendo hecho esto usted habrá convertido un número binario de 8 bits en un número de dos dígitos en código hexadecimal. Cada grupo de cuatro dígitos binarios se convierte independientemente del otro.

Como un ejemplo, considere el número binario de 8 bits,

1 0 0 1 1 1 0 1

Primero, separe este número binario en dos grupos de cuatro dígitos binarios cada uno

1001 1101

Finalmente, substituir el dígito equivalente hexadecimal para cada uno de estos dos grupos

9 D

Esta es la respuesta correcta, 9D (16) significa "relativo al" sistema hexadecimal. A continuación se dan unos números hexadecimales y sus correspondientes números binarios de 8 bits:

Número decimal	Número binario	Número hex
64	0100 0000	40
72	0100 1000	48
73	0100 1001	49
74	0100 1010	4A
96	0110 0000	60
120	0111 1000	78
127	0111 1111	7F
128	1000 0000	80
160	1010 0000	A0
184	1011 1000	B8
191	1011 1111	BF
248	1111 1000	F8
255	1111 1111	FF

UNA NOTA ACERCA DE LA NOTACION

Puede haberle ocurrido que al tratar de todos estos métodos diferentes de representación de números –binario, hex y decimal–, se produzca alguna confusión. Por ejemplo el número 10 puede ser decimal o hex o un número binario. Para remediar este problema, siempre que exista alguna posibilidad para la ambigüedad, todos los números hexadecimales estarán seguidos por la letra H, por ej. 10H, todos los números decimales estarán seguidos por un punto decimal por ej. 10., y todos los números binarios aparecerán sin ninguna notación especial, por ej. 10 ó 0110.

DEMOSTRACIONES

En los primeros tres capítulos hemos incluido una colección de ejercicios que hemos llamado demostraciones. Estas demostraciones están destinadas a animarle a hacer funcionar el Nanocomputador inmediatamente, aunque usted no pueda comprender completamente el Nanocomputador por ahora. Es importante que usted trabaje con estas demostraciones aunque usted piense algunas veces que solamente está pulsando botones y no comprenda lo que está sucediendo.

DEMOSTRACION N.° 1

Paso 1

Refiriéndose al Manual de Instrucción del Nanocomputador, aplique la tensión a su Nanocomputador. Pulse la tecla RESET. Se encenderán varios números de siete segmentos. Si no, pulse de nuevo RESET. Si después de apretar varias veces el RESET no se pone en marcha su Nanocomputador, usted tiene un problema.

Paso 2

Observe que el teclado del Nanocomputador tiene dos teclas con flechas encima de ellas. Apriete una de estas teclas varias veces y observe lo que sucede.

Observaremos tres cosas. Primero observamos que la lámpara roja de selección efectúa un ciclo de once posiciones distintas. También observamos que la lámpara roja se puede mover de un paso cada vez apretando y soltando rápidamente la tecla; por otra parte, se puede hacer que la lámpara roja recorra el ciclo automáticamente manteniendo apretada esta tecla y soltándola cuando la lámpara de selección alcance la posición deseada. Finalmente, observamos que los dígitos que aparecen en el display rojo cambian de acuerdo con la posición de la lámpara roja de selección.

Paso 3

Apriete la otra llave que está marcada con una flecha y observe que es lo que sucede.

Observamos que la lámpara roja de selección efectúa un ciclo entre once posiciones distintas en el sentido opuesto del que se ha observado en el paso 1.

Paso 4

Posicione la lámpara de selección en la posición que indica MEM. Observe lo que aparece en los cuatro dígitos que están más a la izquierda.

Observamos 0000.

Paso 5

Apriete la tecla que pone INC varias veces y observe lo que sucede.

Observamos que aparece la siguiente secuencia de dígitos en el display rojo que está más a la izquierda:

0000 0001 0002 0003 0004 0005 0006 0007 0008 0009
000A 000b 000C 000d 000E 000F 0010 0011 0012 0013

y así sucesivamente.

Obsérvese que la secuencia de dígitos hexadecimales 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F sale al display, con el dígito menos significativo a la derecha en cada grupo de cuatro dígitos. Obsérvese también que los dígitos hexadecimales A, C, E y F aparecen como letras mayúsculas, pero que los dígitos hexadecimales b y d aparecen como letras minúsculas. Esto es simplemente un truco del display de siete segmentos que se utiliza para representar la letra, y así se representará por la B y la D respectivamente. El número hexadecimal de cuatro dígitos del 0000 al 000F representa los dígitos decimales del 0 al 15, el dígito hex 0010 representa al número decimal 16, 0011 representa 17 y así sucesivamente. Así nosotros tenemos un display hexadecimal. La conclusión es que el Nanocomputador se comunicará con nosotros utilizando la representación hexadecimal de los números, de forma que tiene sentido el que nosotros hablemos con el Nanocomputador utilizando la misma representación hexadecimal (hex).

Paso 6

Pulse el botón de RESET. Obsérvese que la lámpara de selección se ha movido. Mueva hacia atrás la lámpara de selección a la posición en donde pone MEM. Tenemos de nuevo en el display 0000, de forma que si pulsamos la tecla INC haremos que el Nanocomputador empiece mostrándonos números hexadecimales sucesivos. ¿Cuál es el mayor número que el Nanocomputador es capaz de mostrarnos utilizando solamente estos cuatro números hex? ¿Cuál es el equivalente decimal de este número?

Respuesta: El mayor número hex que puede mostrar el Nanocomputador es FFFF. El número decimal equivalente a este número es 65.535.

REPASO

Las siguientes preguntas le ayudarán a revisar los códigos digitales.

1. ¿Qué es un código digital?
2. Escriba varios tipos diferentes de códigos digitales.

3. ¿Cuántos bits hay en los siguientes números binarios?
   a. 11010011
   b. 1000000000000011
   c. 1001
4. ¿A qué números decimales corresponden los siguientes números binarios?
   a. 11101
   b. 11111111
   c. 1111111111111111
   d. 1001
   e. 11010011
   f. 10011
5. ¿A qué números hexadecimales corresponden los siguientes números binarios?
   a. 11010011
   b. 00111110
   c. 01110110
   d. 00111100
   e. 11111111
   f. 00110010
   g. 11000011
   h. 00000010
   i. 110
6. ¿A qué números binarios corresponden los siguientes números hexadecimales?
   a. D3H
   b. FFH
   c. 32H
   d. 3EH
   e. 76H
   f. 02H
   g. 5H
   h. 3CH
   i. 00H
7. ¿Qué significan los siguientes subíndices?
   a. (16)
   b. (10)
   c. (2)
8. Defina los siguientes términos.
   a. sistema de contaje hexadecimal
   b. bit
   c. código binario
   d. comunicación
   e. lenguaje
9. ¿Qué son los siguientes números: binario, hex o decimal?
   a. 1111
   b. 1101.
   c. 1100H

RESPUESTAS

1. Un código digital es un sistema de símbolos que representan valores de datos y constituyen un lenguaje especial que pueden entender un computador o un circuito digital.
2. Código binario. Binario codificado en decimal. Código de Gray. Código ASCII. Código EBCDIC. Código Baudout. Código de instrucción del IBM 370. Código de instrucción del Z-80.

3. a. Ocho
   b. Dieciséis
   c. Cuatro
4. a. 29.
   b. 255.
   c. 65,535.
   d. 9.
   e. 211.
   f. 19.
5. a. D3H
   b. 3EH
   c. 76H
   d. 3CH
   e. FFH
   f. 32H
   g. C3H
   h. 02H
   i. 06H
6. a. 11010011
   b. 11111111
   c. 00110010
   d. 00111110
   e. 01110110
   f. 00000010
   g. 101
   h. 00111100
   i. 00000000
7. a. Se refiere al sistema de contaje hexadecimal
   b. Se refiere al sistema decimal
   c. Se refiere al sistema binario
8. a. Un sistema de contaje que está basado en una base de 16.
   b. Una unidad elemental de información que es igual a una decisión binaria, o a la designación de uno de dos posibles estados de cualquier elemento que se utilice para guardar o transmitir información.
   c. Un código en el cual cada elemento del mismo está en uno de dos posibles estados distintos, los cuales son comúnmente conocidos como lógico 0 y lógico 1.
   d. Impartir, conducir, o intercambiar ideas, conocimientos, información, etc. (bien sea mediante la palabra, escritura, o signos).
   e. El conjunto completo de palabras y de métodos o combinación de palabras utilizadas por una nación, pueblo o raza.
9. a. binario
   b. decimal
   c. hexadecimal.

2

Una introducción a la programación de los microcomputadores

En los capítulos que seguirán, usted realizará dos clases diferentes de experimentos: (a) experimentos que solamente necesitan programación del microcomputador, y (b) experimentos que necesitan programación e *interface* del microcomputador, entendiéndose por interface, el cableado de circuitos que conectan el microcomputador a algún dispositivo externo. Puesto que un común denominador de todos los experimentos es la programación, deseamos en primer lugar introducirle en los principios básicos de la programación y en las características del *lenguaje de programación* que utilizaremos en este texto: el conjunto de instrucciones para el microprocesador Z-80. Al mismo tiempo, definiremos una variedad de términos importantes, incluyendo *computador, lenguaje mnemónico, lenguaje máquina* y muchos otros. Esta introducción a la programación ocupará doce capítulos. Preferimos darle las nuevas instrucciones de programación en grupos de cinco a diez, en lugar de todas ellas a la vez.

OBJETIVOS

Al final de este capítulo, usted será capaz de hacer lo siguiente:

- Definir *computador digital.*
- Definir el *microcomputador.*

- Distinguir entre las instrucciones del microcomputador escritas en código binario, código hex o código mnemónico.
- Distinguir entre representación mnemónica y lenguaje máquina.
- Definir *byte.*
- Convertir una dirección de 16 bits en los bytes de dirección alta HI y baja LO.
- Convertir instrucciones de ocho bits codificadas en binario a instrucciones codificadas en hex y viceversa.
- Distinguir entre memoria de lectura/escritura y memoria de sólo lectura.
- Definir *memoria.*
- Definir *programa del computador.*
- Dar la gama de posiciones de memoria, en código binario o hex, para su microcomputador.
- Identificar bytes de 8 bits en una lista de números binarios.

¿QUE ES UN COMPUTADOR?

Existen muchos diferentes tipos de computadores en el mundo: COMPUTADORES DIGITALES, COMPUTADORES ANALOGICOS, COMPUTADORES FLUIDICOS, COMPUTADORES MECANICOS. En este libro usted solamente estudiará los COMPUTADORES DIGITALES, los cuales comprenden probablemente el 99% de todos los computadores utilizados hoy en día. Un computador digital puede definirse como sigue:

Computador digital:

- Un dispositivo electrónico que es capaz de aceptar, guardar y manipular en forma aritmética la información, la cual incluye los datos y el programa de control. La información es tratada en forma de dígitos binarios codificados (0 y 1) que son representados por dos niveles de tensión.[4]
- Cualquier dispositivo, normalmente electrónico, capaz de aceptar información, comparar, sumar, restar, multiplicar, dividir e integrar esta información, que está en forma de dígitos binarios codificados (0 y 1), y proporcionar entonces los resultados de estos procesos en una forma aceptable. Los elementos principales de un computador digital normalmente incluyen la memoria, el control, la unidad aritmética y lógica, y los dispositivos de entrada y salida.[2]

Se debe remarcar que un computador digital manipula *información binaria,* del

tipo que hemos discutido en el capítulo 1. La información binaria está generalmente en la forma de *códigos digitales:* códigos de instrucción; códigos utilizados para representar números decimales en forma digital; códigos utilizados por los circuitos electrónicos para realizar varias operaciones digitales; y códigos utilizados para representar en forma digital el alfabeto, números decimales, símbolos y otras operaciones.

¿QUE ES UN MICROCOMPUTADOR?

Un *microcomputador* es un computador digital completamente operacional que está basado en un *circuito integrado* (*chip*) microprocesador. Un microprocesador es un solo *circuito integrado* que posee por lo menos el 75% de la potencia de cálculo y manipulación de datos de un computador digital. Normalmente no puede funcionar sin la ayuda del soporte de chips auxiliares y de memoria.

Un *chip de circuito integrado* es un dispositivo electrónico en el cual los elementos activos (transistores) y pasivos (resistencias) están contenidos en un solo encapsulado. En la electrónica digital, el término se aplica principalmente a circuitos que contienen elementos de semiconductor.[2] El chip del microprocesador es un producto de tecnología avanzada de la industria de los semiconductores, nacido básicamente de la capacidad que tienen ahora los fabricantes de colocar miles de transistores en un solo chip de silicio no mayor de 60 a 80 mm^2.

¿QUE ES UN PROGRAMA DE COMPUTADOR?

Un *programa de computador* puede definirse como una serie de instrucciones o sentencias preparadas en una forma aceptable para el computador, cuyo propósito es alcanzar en cierto resultado.[2] Esta definición no implica cual debe ser el resultado deseado. Por ejemplo, usted puede estar simplemente interesado en arreglar los datos de entrada digitales de una forma más conveniente, la cual puede ser almacenada o proporcionada como una salida. Con los microcomputadores, usted estará más interesado en escribir programas para el microcomputador que controlen el funcionamiento de un dispositivo o máquina. En una lavadora casera usted puede desear controlar la cantidad de agua utilizada, la temperatura del agua para diferentes ciclos de lavado, el número y clases de ciclos utilizados para lavar un determinado tipo de fibra y el tiempo de duración de cada ciclo. Todo esto puede hacerse con un programa de computador escrito en forma adecuada.

INSTRUCCIONES

Una *instrucción* de computador puede ser definida como un grupo de caracteres que definen una operación. Bien sea sola o con otra información, una instrucción hace que un computador digital realice la operación de manipular las cantidades indicadas.

Un *carácter* es un símbolo de un conjunto de símbolos elementales, tales como los que corresponden a las teclas de una máquina de escribir. Los símbolos normalmente incluyen los dígitos decimales del 0 al 9, las letras de la A a la Z, los puntos, signos de dólar, comas, símbolos de operación, y otros símbolos solos que el computador puede leer, guardar o escribir.[5] En un programa para un computador es bastante común que se utilicen todos los símbolos del teclado, incluyendo símbolos como @, #, $, %, &, *, (,), /, y posiblemente otros.

Las instrucciones de un computador pueden expresarse de muy variadas formas. Pueden expresarse como números binarios,

```
11010011
00111110
```

números hex,

```
D3H
3EH
```

código mnemónico,

```
OUT 3EH
LD A,02H
```

palabras completas,

SACAR LOS DATOS DE ACUMULADOR AL DISPOSITIVO # 3E (HEX)
CARGAR EL DATO 02 HEX EN EL REGISTRO A

o expresiones matemáticas completas,

X = A**2 + B*y + C

En este libro expresaremos las instrucciones al nivel de números binarios, números hex y representaciones mediante mnemónicos.

MNEMONICOS

Mnemónico es un término que describe algo que se utiliza para ayudar a la memoria humana. En vista de esta definición, tenemos lo siguiente:

código mnemónico: Instrucciones del computador escritas en una forma que el

programador pueda recordar fácilmente, pero que se deben convertir a lenguaje máquina más adelante, bien sea mediante el computador o por el usuario.[2]

lenguaje mnemónico: Un lenguaje de programación que se basa en símbolos que se pueden recordar fácilmente y que pueden ser ensamblados en lenguaje máquina mediante un computador.[2]

operación mnemónica: Instrucciones de un computador que están escritas en una notación significativa, por ejemplo, ADD, LD y OUT.[2]

INSTRUCCIONES

En esta serie de capítulos, utilizaremos ocasionalmente los códigos mnemónicos para las instrucciones que usted utilizará cuando programe el microcomputador. Los códigos mnemónicos serán los sugeridos por ZILOG Corporation para el conjunto de instrucciones de su microprocesador Z-80, que contiene 158 tipos diferentes de instrucciones máquina. Con el tiempo usted será capaz de convertir rápidamente desde lenguaje máquina (por ejemplo código binario) a código mnemónico, y viceversa.

LENGUAJE MAQUINA

El moderno computador digital electrónico es capaz de realizar manipulaciones utilizando señales electrónicas binarias, típicamente dos niveles de tensión (+5 volt y el potencial de masa) que representan los estados lógicos 1 y 0, respectivamente. Así cada instrucción del computador está escrita como una serie de 1 y 0 que específicamente caracterizan a esta instrucción y no a otra. A esta representación binaria de las instrucciones de un computador se le llama *lenguaje de máquina* o *código máquina.* Por ejemplo, la instrucción en lenguaje máquina 00000111 desplaza el contenido del acumulador del microprocesador Z-80 un bit hacia la izquierda. La instrucción, 00001111, desplaza el contenido del acumulador un bit hacia la derecha.

En esta serie de capítulos, usted se habituará al uso de las instrucciones en lenguaje máquina para el microprocesador Z-80. Se le darán las instrucciones en CODIGO HEX de forma que las pueda recordar fácilmente. Algunos códigos de instrucción, que usted utilizará pronto, para los experimentos simples de programación del microcomputador incluyen:

C3H Instrucción de salto incondicional
76H Instrucción de paro
3CH Incrementar el contenido del acumulador de 1
3EH Instrucción de carga del acumulador en forma inmediata

Todas estas nuevas frases que hemos utilizado en esta sección –salto incondicional, carga, paro, incremento, etc.– se discutirán brevemente.

UN PROGRAMA SIMPLE

Vamos a examinar el siguiente programa para el Z-80.

00H	No operación
3EH	Cargar el contenido del próximo byte del programa en el acumulador
FFH	Byte de datos
76H	Paro

Este programa contiene tres instrucciones y un byte de datos. En este caso se ha escrito el programa en el código hex, que hemos estudiado en el capítulo 1. También se habría podido escribir el mismo programa en código binario, como se muestra a continuación.

00000000	No operación
00111110	Cargar el contenido del próximo byte del programa en el acumulador
11111111	Byte de datos
01110110	Paro

Alternativamente, se habría podido escribir en código mnemónico y más tarde convertirlo a código máquina con la ayuda de un programa especial llamado *ensamblador.* Así, tenemos el siguiente programa en código mnemónico:

NOP	No operación
LD A,FFH	Cargar el byte de datos FF en el acumulador
HALT	Paro

Obsérvese que el programa en código mnemónico consiste principalmente en palabras o abreviaciones de palabras, tales como NOP, HALT y LD.

¿Cómo ejecuta el microcomputador este programa? Lo hace paso a paso, ejecutándose en primer lugar la primera instrucción, NOP. Se produce la siguiente secuencia de operaciones:

1. El microcomputador ejecuta la instrucción NOP, que hace que el microcomputador haga una "pausa" durante un *ciclo de instrucción.* El computador avanza entonces a la siguiente instrucción. Existe una utilización importante de la instrucción NOP que usted verá en un programa posterior.
2. Cuando el microcomputador ejecuta la instrucción LD, que tiene como

código 3E hex, mira a la posición de memoria que sigue para determinar el valor que deberá cargar en el acumulador.

3. El microcomputador va a la próxima posición de memoria, en donde encuentra FF. El toma este valor y lo almacena en el acumulador del microcomputador.
4. El microcomputador ejecuta la instrucción final, HALT. Esta hace que el computador se pare.

El programa anterior puede parecer sencillo, o puede que no. ¿Qué es *memoria*? ¿Qué es un *byte*? ¿Cómo se puede distinguir entre una *instrucción* y un *byte de datos*? ¿Qué es el acumulador? Todas estas preguntas son muy razonables, algunas de las cuales usted se puede haber preguntado a sí mismo cuando ha estudiado el programa anterior.

Vamos a proceder a responder a algunas de estas preguntas.

BYTE

Un *byte* es un grupo de ocho bits contiguos que ocupan una sola posición de memoria en un microcomputador que esté basado en el Z-80. Por "contiguos" queremos decir adyacentes o próximos, o uno después del otro. Un byte puede ser cualquiera de las 256 combinaciones posibles de ocho dígitos binarios cada uno de los cuales puede ser o bien 0 ó 1. La única restricción es que un byte contiene exactamente ocho bits. Así, el número binario,

0 1 1 0 1 0 0 1

es un byte, mientras que el número binario,

1 0 1 0 0 1

no es un byte puesto que sólo contiene seis bits. El término byte se ha popularizado debido a que muchos computadores digitales tienen longitudes de palabra que son múltiples de ocho bits. Para referenciar fácilmente los bits contenidos en un byte, éstos se numeran del 0 al 7:

D7 D6 D5 D4 D3 D2 D1 D0

La "D" (probablemente abreviación de Dato) algunas veces no está presente. El *bit más significativo* (MSB) es el D7. El *bit menos significativo* (LSB) es D0.

En general, una *palabra* es el número de bits que puede manipular un computador simultáneamente. Si el número de bits en una *palabra* es de ocho, normalmente utilizamos el término byte en lugar de palabra. De cualquier forma el Z-80 tiene una longitud de *palabra* de 8 bits. La longitud de palabra en un minicomputador PDP 8 es de 12 bits, lo que significa que el minicomputador PDP 8 manipula

12 bits a la vez cuando está ejecutando un programa. El minicomputador PDP 11 tiene una longitud de palabra de dieciséis bits, y los computadores mayores generalmente tienen una longitud de palabra de 32 bits, 36 bits o 60 bits.

MEMORIA

La *memoria* puede ser definida como cualquier dispositivo que pueda guardar los estados lógicos 1 y 0 de tal forma que se pueda acceder o almacenar un solo bit o grupo de bits.[6] Existen muchos tipos diferentes de memoria que satisfacen este requerimiento; sin embargo en su microcomputador, usted tiene solamente dos clases distintas de memoria:

memoria de lectura/escritura: Una memoria de semiconductor en la cual se pueden escribir (almacenar) y leer de nuevo (recuperar) los estados lógicos 0 y 1.[6] A estas memorias también se las llama *memorias de acceso aleatorio* (RAM) (**R**andom **A**ccess **M**emories).

memoria de sólo lectura: Una memoria de semiconductor desde la cual se pueden leer los datos muchas veces pero en las que no se puede escribir como en el caso de las memorias de lectura/escritura.[6] En forma abreviada se las llama ROM (**R**ead **O**nly **M**emory).

Actualmente, la memoria de sólo lectura en su microcomputador puede ser de un tipo especial llamada *memoria de sólo lectura que se puede borrar y programar,* o EPROM (**E**rasable **P**rogrammable **R**ead **O**nly **M**emory). Hablaremos acerca de las EPROM en un próximo capítulo.

Lo importante es que su memoria consta de dispositivos de semiconductor. Son rápidas y relativamente baratas, no tienen partes mecánicas, y no ocupan mucho espacio en la tarjeta de circuito impreso. Constituyen una de las razones por las que la tecnología de los computadores ha avanzado tan deprisa como lo ha hecho.

¿De cuánta memoria dispone usted? El Nanocomputador más sencillo que usted utiliza contiene 4096 bytes de memoria de lectura/escritura y de 2048 bytes de memoria de sólo lectura. El programa que le permite entrar datos en el teclado y muestra la información en un display de siete segmentos está cargado en los 2048 bytes de memoria de sólo lectura. Nos referiremos a este programa como el sistema operativo del Nanocomputador.

Puesto que un byte contiene ocho bits, esto significa que usted tiene por lo menos un total de 49.152 bits de memoria en su microcomputador. Esto es suficiente para todos los experimentos de programación e interface que usted encontrará en este libro.

En el segundo nivel del Nanocomputador usted será capaz de incrementar el número de bytes de memoria de lectura/escritura a 16.384 e incrementar el número de bytes de memoria de sólo lectura a 8192. Un Nanocomputador con esta capacidad de memoria tiene un total de 24.576 bytes. Es conocido como un microcomputador de 24K, en donde "K" representa aproximadamente mil (exactamente 1024) posiciones distintas de memoria. Un microcomputador de 4K contendrá 4096 bytes de memoria. Un microcomputador de 64K contendrá 65.536 bytes de memoria.

DIRECCION DE MEMORIA

La *dirección de memoria* se define como la posición de almacenamiento de una palabra de memoria. Nótese que decimos palabra, no byte. Para algunos computadores, una palabra puede contener 32 bits, de forma que cada posición distinta de memoria contendrá 32 bits. Para el microcomputador Z-80, cada posición de memoria contiene un solo byte, es decir ocho bits.

Con el Nanocomputador estándar que usted puede utilizar hay 6144 posiciones de memoria distintas. Estas posiciones de memoria están divididas en dos grupos, que se pueden describir de la siguiente forma:

- Memoria grupo 1: El primer grupo de 4096 (4K) posiciones de memoria, de ocho bits cada una. Esta es la memoria de lectura/escritura que usted utilizará normalmente cuando usted programe su microcomputador.
- Memoria grupo 2: El segundo grupo de 2048 (2K) posiciones de memoria, conteniendo cada una 8 bits. Esta región de la memoria está ocupada por memoria de sólo lectura, o tal vez por memoria de sólo lectura, borrable y programable EPROM, la cual contiene el sistema operativo del Nanocomputador que le permite funcionar. USTED NO PUEDE CAMBIAR EL CONTENIDO DE ESTE GRUPO DE MEMORIA.

GAMA DE POSICIONES DE MEMORIA

El chip del microprocesador Z-80 es muy notable, puede direccionar 65.536 (64K) posiciones de memoria distintas, conteniendo cada una de ellas 8 bits. El chip contiene 16 bits de palabra de direccionamiento. Si usted realiza un cálculo simple obtendrá que 2 elevado a la potencia 16 (2**16) equivale a 65.536.

Como se ha indicado anteriormente, su Nanocomputador puede tener solamente 6144 (6K) posiciones de memoria disponibles en la tarjeta básica. Usted se puede preguntar, ¿qué posiciones son las utilizadas entre las posibles 65.536?

Nuestra respuesta: las primeras 4K posiciones juntamente con los últimos 2K de la memoria.

Dicho de otro modo, las direcciones de la memoria que se pueden utilizar en el Nanocomputador estándar son:

0000000000000000 (BASE 2) a 0000111111111111 (BASE 2) Memoria R/W
1111100000000000 (BASE 2) a 1111111111111111 (BASE 2) ROM

Esta es una notación difícil y resulta muy difícil de recordar. Existe una forma más fácil de identificar las posiciones de memoria y la gama de su microcomputador. Esto se discutirá en la próxima sección.

DIRECCIONES DE MEMORIA HI Y LO

Resulta difícil recordar una dirección de memoria de 16 bits, bastante más que un código de instrucción de 8 bits o byte de datos. El chip del microprocesador Z-80 trata las direcciones de memoria de 16 bits como dos bytes de dirección de memoria, un byte HI de 8 bits y un byte LO de 8 bits. Estos se definen como sigue:

HI byte de dirección: Los ocho bits más significativos (o los que están más a la izquierda) en una palabra de dirección de memoria de 16 bits para el chip del microprocesador Z-80. En forma abreviada H o HI (de HIGH).

LO byte de dirección: Los ocho bits menos significativos (o los que están más a la derecha) en una palabra de dirección de memoria para el chip del microprocesador Z-80. En forma abreviada L o LO (de LOW).

Así, la posible gama de posiciones de memoria de lectura/escritura para su microcomputador es

HI = 00000000 (2) HI = 00001111 (2)
a
LO = 00000000 (2) LO = 11111111 (2)

Recuerde que usted ha aprendido a convertir un número binario de 8 bits a un número hex de 2 dígitos. Aplicándolo a las direcciones de memoria HI y LO anteriores, usted debe obtener la siguiente gama de posiciones de memoria de lectura/escritura:

HI = 00 (16) HI = 0F (16)
a
LO = 00 (16) LO = FF (16)

Recuerde la siguiente regla: Para especificar una posición de memoria se debe especificar los dos bytes de dirección HI y LO, que juntos expresan una palabra de dirección de memoria de 16 bits.

DEMOSTRACION N.° 1

A cada paso en esta demostración, usted debe tener la lámpara de selección situada en la posición MEM. Obsérvese en la figura 2-1 que existen cuatro dígitos hex encendidos en la parte de la izquierda de los displays en rojo, y dos dígitos hex en la parte de la derecha.

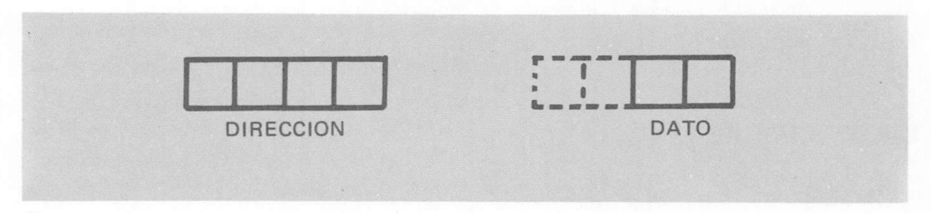


Fig. 2-1. – Demostración de la posición de memoria.

Los cuatro dígitos hex de la izquierda representan la dirección de una posición de memoria. Los dos dígitos hex que se muestran a la derecha representan el contenido de la posición de memoria cuya dirección muestran los dígitos de la izquierda. Podemos pensar que la memoria está constituida por una colección de cajas. Cada caja tiene una etiqueta impresa permanentemente en ella. Estas etiquetas son actualmente números hexadecimales empezando con 0000, 0001, 0002, 0003, y así sucesivamente. Dentro de cada caja usted puede poner exactamente un byte de información en forma de dos dígitos hex. Utilizando el teclado del Nanocomputador usted puede examinar posiciones de memoria individuales y cambiar el contenido de una determinada posición de memoria.

Paso 1

Colocar la lámpara de selección en MEM. Observamos que la dirección de memoria, que se muestra en los cuatro dígitos de la izquierda, era 0000. El contenido de la posición de memoria 0000, que se muestra en los dos dígitos de la derecha, llamado el *display de datos,* era 00.

Vamos a cambiar el contenido de 0000H de 00H a 23H. Pulsar la tecla del 2, y a continuación la tecla hex 3, y finalmente pulsar la tecla denominada ST. ST es una abreviación de la palabra STORE (almacenar). Al pulsar ST usted guardará el número 23H en la posición 0000H. Obsérvese que ahora la dirección se ha incrementado automáticamente a 0001H, de forma que está "señalando" a la posición de memoria 0001H. Guarde 24H en esta posición.

Paso 2

Examine ahora las posiciones de memoria 0000H y 0001H para verificar que se ha almacenado efectivamente 23H y 24H en las mismas. Pulsar 0, 0, 0, 0 secuencialmente en el teclado y a continuación en la tecla denominada LA. LA es una abreviación de LOAD ADDRESS (cargar la dirección). Usted está cargando la dirección hex 0000 en el display de direcciones. Ahora podemos observar que en el display de la derecha aparece 23 como el dato contenido en la posición de memoria 0000H. Pulse la tecla denominada INC. Obsérvese que la dirección de memoria se ve incrementada de 1 y que el contenido de esta posición de memoria es efectivamente 24H. Usted puede ahora ser capaz de determinar el contenido de cualquier posición de memoria y de cambiar el contenido de cualquier posición de memoria de lectura/escritura a cualquier valor que usted desee.

Paso 3

Usted observará ahora el contenido de una posición de memoria ROM (memoria de sólo lectura) e intente escribir en ella. Mire al contenido de la posición de memoria FC00H (pulsando F, C, 0, 0 en orden y pulsando LA). Observamos que el contenido de FC00H es 31H. Intente cargar el valor FFH en esta posición de memoria (pulsar F, F y pulsar de nuevo la tecla ST). Examine ahora el contenido de FC00H de nuevo. Observamos que el contenido de FC00H no ha cambiado, y que todavía es 31H. Así vemos que NO hemos sido capaces de *escribir* en la memoria de sólo lectura.

REPASO

1. Identificar las siguientes instrucciones diciendo en que tipo de código están representadas, código binario, hex, o código mnemónico.
   a. HALT
   b. 11010011
   c. 3E
   d. LD
   e. INC
   f. 00111100
   g. 76
2. Escribir las siguientes instrucciones binarias en código hex.
   a. 11010011
   b. 01110110
   c. 00111100
   d. 00110010
   e. 00000000

f. 11000011
g. 11111111

3. ¿Cuál de estos números es un byte?
   a. 1001
   b. 011
   c. 0000001100000011
   d. 1110001101
   e. 111000
   f. 0100110
4. Escribir la siguiente dirección de memoria de 16 bits en forma de los dos bytes hex, HI y LO.
   a. 0000001111111111
   b. 0000000011111111
   c. 0000000111111111
   d. 0000001011111111
   e. 0000000000000000
   f. 0000000100000000
   g. 0000001000000000
   h. 0000001100000000
5. ¿Cuáles de las siguientes instrucciones están en lenguaje máquina?
   a. NOP
   b. HALT
   c. LD
   d. INC
   e. 3EH
   f. 76H
   g. 11010011
   h. 00H
   i. 00111100
6. Escribir la gama de memoria de los siguientes grupos de memoria, en términos de byte de dirección HI y LO, en el microcomputador Z-80.
   a. Los primeros 4K byte de memoria (lectura/escritura)
   b. Los primeros 16K de memoria (lectura/escritura)
   c. Los últimos 4K byte de memoria (sólo lectura)
   d. Los últimos 8K byte de memoria (sólo lectura)
7. Defina los siguientes términos.
   a. byte
   b. dirección de memoria
   c. código mnemónico.

RESPUESTAS

1. a. código mnemónico
   b. código binario
   c. código hex
   d. código mnemónico
   e. código mnemónico
   f. código binario
   g. código hex
2. a. D3
   b. 76
   c. 3C
   d. 32

e. 00
f. C3
g. FF

3. Ninguno de los ejemplos es un byte. Un byte debe contener exactamente 8 bits.

4. a. HI=03 LO=FF
   b. HI=00 LO=FF
   c. HI=01 LO=FF
   d. HI=02 LO=FF
   e. HI=00 LO=00
   f. HI=01 LO=00
   g. HI=02 LO=00
   h. HI=03 LO=00

5. Los ejemplos g e i están en lenguaje máquina.
6. a. La gama es HI = 00 y LO = 00 hasta HI = 0F y LO = FF
   b. La gama es HI = 00 y LO = 00 hasta HI = 4F y LO = FF
   c. La gama es HI = F0 y LO = 00 hasta HI = FF y LO = FF
   d. La gama es HI = E0 y LO = 00 hasta HI = FF y LO = FF
7. a. Un grupo de ocho bits contiguos que ocupan una sola posición de memoria en un microcomputador Z-80.
   b. La posición de almacenamiento de una palabra de memoria.
   c. Instrucciones de un computador escritas en una forma que el programador pueda recordar fácilmente, pero que deben ser convertidas en lenguaje máquina, más adelante, para que estén en una forma que el computador pueda leer.

3

Algunas instrucciones de la CPU del microprocesador Z-80

En este capítulo definiremos varios términos importantes, incluyendo *operación, byte de datos, byte de dirección* y *código de dispositivo.* También le introduciremos a varias instrucciones simples del microprocesador Z-80 que usted utilizará en los programas proporcionados en el capítulo 5. Nuestro objetivo es el de introducirle gradualmente al conjunto de instrucciones del Z-80 y proporcionar programas que le permitan ver como se utilizan algunas instrucciones básicas.

OBJETIVOS

Al final de este capítulo, usted será capaz de lo siguiente:

- Definir qué es un *programa de computador.*
- Definir *operación.*
- Proporcionar representaciones simples para las instrucciones de un solo byte, dos bytes, tres bytes y cuatro bytes.
- Explicar las diferencias entre las siguientes clases de bytes de programa: código de operación, byte de datos, código del dispositivo, byte de dirección HI, byte de dirección LO y byte de desplazamiento.
- Definir *registro.*
- Escribir los dos grupos de seis registros de uso general y los seis registros de uso especial en el chip del microprocesador Z-80.

- Escribir cuales son los registros de uso general que se utilizan como pares de registros.
- Definir *acumulador.*
- Definir *incremento.*
- Explicar el funcionamiento de cinco instrucciones corrientes del microcomputador Z-80: NOP, HALT, INC A, LD A, dato y JP dirección.
- Definir el *modo de direccionamiento inmediato.*
- Para un microcomputador Z-80 de 2,5 MHz, escribir los tiempos de ejecución de las siguientes instrucciones del microcomputador: NOP, HALT, INC A, LD A, datos y JP dirección.

¿QUE ES UN PROGRAMA DE COMPUTADOR?

Un *programa de computador* se puede definir como una secuencia de instrucciones que tomadas como un conjunto, permiten realizar al computador una secuencia de operaciones para llevar a término una determinada tarea. ¿Cuál es la tarea? Puede ser cualquier cosa que entre dentro de las capacidades del computador, los dispositivos externos de entrada/salida asociados y la memoria.

Los programas se guardan en la memoria como una secuencia de 0 y 1 (bits), que el computador puede leer, interpretar y ejecutar en secuencia, una cada vez. Para el Z-80 estos bits se guardan en grupos de 8 bits, llamados bytes. Una sola instrucción puede ocupar uno, dos, tres o cuatro bytes consecutivos en la memoria. El Z-80 ejecuta un programa, leyendo una instrucción, interpretando los grupos de bits, y realizando entonces las tareas necesarias para completar la operación definida por la instrucción. Se leen las posiciones consecutivas de la memoria hasta que se alcanza una instrucción que le dice al computador que se pare o que salte a otra posición de memoria para la siguiente instrucción.

Los programas no incluyen solamente bytes con instrucciones. También se deben incluir bytes de datos en los programas para proporcionar la información necesaria. Por ejemplo un programa diseñado para sumar dos números debe incluir los números que se han de sumar (byte de datos) así como las instrucciones para realizar la operación de suma (bytes de instrucción). Otros tipos de bytes que forman parte de un programa incluyen los bytes de dirección, los bytes de código del dispositivo y los bytes de desplazamiento. Estos se discutirán más adelante en este capítulo.

La configuración mínima del Nanocomputador proporciona al usuario 4K bytes de memoria de lectura/escritura para el almacenamiento del programa del usuario. Esto es suficiente para guardar programas muy complejos. Dos términos muy críticos en nuestra definición de los programas de un computador son "instrucción" y "operación". Vamos ahora a investigar su significado más a fondo.

INSTRUCCIONES Y OPERACIONES

Una *instrucción* es un conjunto de caracteres que definen una operación, sola o asociada a otra información, y que juntas hacen que el computador realice la operación. Una *operación* se define como una acción específica que el computador realiza cuando una instrucción lo necesita (por ejemplo, división, suma, resta, función lógica O, etc.) El número de operaciones distintas que un computador puede realizar y la velocidad con la cual puede realizar estas operaciones proporciona una medida de lo "potente" que es un computador. Las operaciones que el microprocesador Z-80 puede realizar pueden ser subdivididas en los siguientes grupos:

Grupo de transferencia de datos
Grupo aritmético y lógico
Grupo de rotación y desplazamiento
Grupo de manipulación de bits
Grupo de jump (salto), call (llamada) y retorno
Grupo de I/O y control de la máquina.

INSTRUCCIONES MULTIBYTE

Muchas instrucciones en el conjunto de las que comprende el Z-80 solamente necesitan un byte, pero otras necesitan dos, tres, o también cuatro bytes sucesivos antes de que se puedan ejecutar. Llamamos a estas últimas instrucciones, *instrucciones multibyte.* Daremos algunas definiciones:

instrucción de un solo byte: Una instrucción que consiste en ocho bits contiguos que ocupan una sola posición de memoria.

instrucciones de dos, tres o cuatro bytes: Una instrucción cuya información ocupa dos, tres o cuatro posiciones sucesivas de memoria.

El número de bytes necesario para una instrucción están estrechamente relacionados con la complejidad de la instrucción y de la información que necesita. El conjunto de instrucciones del Z-80 fue diseñado como una extensión del conjunto de instrucciones de un microprocesador, el 8080, manufacturado por Intel Corporation. Para mantener la consistencia entre los dos conjuntos de instrucciones, fueron necesarios ciertos compromisos en la definición de las nuevas instrucciones del Z-80. Esto ha tenido como consecuencia que la estructura de las instrucciones del Z-80 sea un poco más complicada que la del 8080. Sin embargo, este

sacrificio está más que compensado por el hecho de que casi todos los programas escritos para el microprocesador 8080 puedan ser ejecutados en un microprocesador Z-80 sin ningún cambio. El microprocesador 8080 es históricamente un chip muy importante, para el cual existe una gran cantidad de software (programas). Así que esta "compatibilidad en exceso" es especialmente beneficiosa.

En los próximos párrafos se darán representaciones simples para las instrucciones de un byte, dos bytes, tres bytes y cuatro bytes. Obsérvese que en todas las instrucciones de cuatro bytes menos una, el primer byte o los dos primeros son códigos de operación que especifican qué es lo que hace la instrucción, y los últimos bytes contienen la información necesaria para llevarla a cabo. Discutiremos esto con mayor detalle cuando introduzcamos instrucciones específicas. Estos formatos de las instrucciones se presentan aquí solamente como un avance de lo que vendrá más adelante. Puesto que los códigos de operación se siguen secuencialmente uno a otro en la memoria, en direcciones numeradas, las escribimos en columnas verticales como en una tabla, al contrario de la página que usted está leyendo que está escrita horizontalmente.

Las instrucciones de un solo byte necesitan solamente un código de operación y no necesitan información auxiliar.

CODIGO DE OPERACION

Las instrucciones de dos bytes tienen cuatro formas:

CODIGO DE OPERACION
CODIGO DE OPERACION

CODIGO DE OPERACION
Byte de dato

CODIGO DE OPERACION
Código de dispositivo

CODIGO DE OPERACION
Byte de desplazamiento

Describiremos brevemente que significan los términos *byte de datos, código del dispositivo* y *byte de desplazamiento.*

Las instrucciones de tres bytes tienen tres formas:

CODIGO DE OPERACION
Byte de dato
Byte de dato

CODIGO DE OPERACION
Byte de dirección LO
Byte de dirección HI

CODIGO DE OPERACION
CODIGO DE OPERACION
Byte de desplazamiento

Hemos discutido previamente los conceptos de bytes de dirección de memoria LO y HI.

Las instrucciones de cuatro bytes tienen cuatro formas:

CODIGO DE OPERACION	CODIGO DE OPERACION	CODIGO DE OPERACION
CODIGO DE OPERACION	CODIGO DE OPERACION	CODIGO DE OPERACION
Byte de datos	Byte de dirección LO	Byte de desplazamiento
Byte de datos	Byte de dirección HI	Byte de dato

CODIGO DE OPERACION
CODIGO DE OPERACION
Byte de desplazamiento
CODIGO DE OPERACION

Como usted bien puede intuir, los dos últimos tipos de instrucciones de cuatro bytes representan instrucciones bastante complicadas. Más tarde se discutirán varios ejemplos de estos tipos de instrucciones.

TIPOS DE INFORMACION GUARDADA EN LA MEMORIA

La memoria en un microcomputador Z-80 consiste en una secuencia de posiciones sucesivas de 8 bits. Siempre que el computador utiliza la memoria lo hace utilizando 8 bits a un tiempo. Existen seis tipos diferentes de información que se pueden guardar en la memoria:

códigos de operación de 8 bits
byte de datos de 8 bits
códigos de dispositivo de 8 bits
bytes de dirección LO de 8 bits
bytes de dirección HI de 8 bits
bytes de desplazamiento de 8 bits

Así, en un programa del Z-80, guardamos simultáneamente códigos de instrucción, bytes de datos, códigos de dispositivo, bytes de dirección y bytes de desplazamiento en la misma memoria. Todo este tipo de información puede existir entremezclada. Es razonable preguntarse como el microcomputador es capaz de distinguir estos distintos tipos de información.

La respuesta básica es que el orden en el que aparece la información determina el tipo de información de que se trata. La programación de un computador es una

actividad de precisión: UN error en el programa y éste no funcionará correctamente. Un programa de microcomputador empieza en una dirección de memoria elegida y procede operación por operación hasta una dirección final de memoria. Los códigos de operación siempre le dicen lo que vendrá después en el programa, es decir, cuando el próximo byte de memoria es un byte de datos, byte de dirección, byte de código de dispositivo, otro código de operación o un byte de desplazamiento.

CODIGO DE OPERACION

El primer byte de una instrucción del Z-80 es siempre un *código de operación.* Téngase en cuenta que algunos tipos de instrucciones empiezan con dos bytes de código de operación. Estas instrucciones son extensiones del antiguo conjunto de instrucciones del 8080. Si el primer byte de una instrucción es CB, DD, ED o FD, entonces el segundo byte debe ser también un código de operación. El(los) byte(s) del código de operación definen la acción específica que realizará el chip del microprocesador Z-80. Las acciones específicas incluyen transferencia de datos, operaciones lógicas, instrucciones de bifurcación, operaciones del stack, operaciones de I/O y operaciones de control de la máquina. Si usted desea conocer lo que el microcomputador va a hacer a continuación, el código de operación de la próxima instrucción se lo dirá. Los sinónimos para el código de operación son: *código op* y *código de instrucción.*

BYTE DE DATOS

Un *byte de datos* se define como un número binario de 8 bits que el chip del microprocesador Z-80 utilizará en una instrucción aritmética o lógica, o para guardarla en la memoria. Los ocho bits pueden ser cualquier clase de código digital: código binario, binario codificado en decimal, código ASCII, etc. Cuando utilizamos el término byte de datos, queremos significar que los ocho bits no son un código de operación, una dirección de memoria, un código de dispositivo o un byte de desplazamiento. Cuando se hace un programa para un microcomputador, es muy conveniente incluir datos en el mismo, donde y cuando los necesite, mejor que tenerse que referir a una posición de memoria remota para los 8 o 18 bits de datos que usted necesita.

CODIGO DE DISPOSITIVO

El *código de dispositivo,* para un microcomputador basado en el Z-80, es lo que

identifica al dispositivo específico de entrada o salida con el cual se desea intercambiar ocho bits de información, y un impulso de selección de dispositivo. Hablaremos de los detalles de como se hace esto más adelante. Lo importante es que el código del dispositivo es un código de 8 bits, lo que significa que se pueden direccionar 2^8, o 256 dispositivos distintos de salida. En su microcomputador, los códigos 04 y 07 para los dispositivos de salida están reservados para el sistema operativo del Nanocomputador.

A medida que usted avance en el estudio de este texto, le recomendamos que estudie detenidamente lo que significa código del dispositivo e *impulso de selección del dispositivo,* y como utilizarlos para obligar a los dispositivos de entrada/salida a que funcionen sincronizados con el programa del microcomputador.

BYTES DE DIRECCION HI Y LO

Deseamos recordarle de nuevo que el *byte de dirección HI* está formado por los 8 bits más significativos, o bits de mayor peso, y que el *byte de dirección LO* está formado por los 8 bits menos significativos, o bits de menor peso, en la palabra de direccionamiento de la memoria del Z-80. Puesto que el Z-80 es un chip microprocesador de 8 bits, que obtiene los datos o instrucciones de la memoria ocho bits cada vez, no hay otra alternativa que tratar la información de la dirección de memoria de 16 bits como un par de bytes de dirección de 8 bits.

BYTE DE DESPLAZAMIENTO

Los bytes de desplazamiento aparecen en las instrucciones que utilizan *Direccionamiento Indexado.* Direccionamiento indexado es una técnica para definir una dirección de memoria de dos bytes añadiendo un *Desplazamiento* a un número de 16 bits que reside en una posición especial del chip del microprocesador llamada *Registro Indice. Un desplazamiento es un número en complemento a dos con signo.*

No intentaremos definir por ahora los números en complemento a dos con signo. Es suficiente decir que es un método para representar números binarios que facilita la manipulación de los números negativos. Esto se explicará cuidadosamente más tarde.

No se sienta desanimado si muchos de los términos anteriores no le resultan familiares. La comprensión completa de estos términos llegará solamente con la experiencia al utilizar las instrucciones para programar su Z-80.

¿QUE ES UN REGISTRO?

Un *registro* es un circuito de almacenamiento a corto plazo cuya capacidad es normalmente de una palabra del computador. Los registros en el microprocesador Z-80 almacenen un solo byte, es decir ocho bits contiguos. Existen varios tipos registros en el Z-80, algunos de los cuales se utilizarán para guardar información digital y otros que son utilizadas por el mismo chip cuando ejecuta las instrucciones. En general, podemos subdividir estos registros del chip en dos grupos distintos: los que se pueden direccionar desde un programa y los que no se pueden direccionar desde un programa. Los registros direccionables por programa incluyen:

- dos grupos de *registros de uso general* de 8 bits direccionados independientemente o por pares,

Grupo 1:		Grupo 2:	
	registro B		registro B'
	registro C		registro C'
	registro D		registro D'
	registro E		registro E'
	registro H		registro H'
	registro L		registro L'

Nos referiremos al grupo 2 como grupo de registros alternativos (ARS) (de Alternate Register Set).

- un ACUMULADOR de 8 bits para cada grupo, también conocidos como registros A y A'.
- un registro de INDICADORES (FLAG) de 8 bits para cada grupo, también conocidos como registros F y F'.
- el registro de 16 bits INDICADOR DEL STACK (SP).
- el REGISTRO CONTADOR DE PROGRAMA de 16 bits (PC).
- dos REGISTROS INDICE de 16 bits (IX) e (IY).
- el registro de 8 bits de DIRECCION DE PAGINA DE INTERRUPCION (I).
- el registro de 8 bits de REFRESCO DE MEMORIA (R).

Estos son los únicos registros con los cuales usted puede intercambiar información directamente con la ayuda de programas del microcomputador escritos adecuadamente.

REGISTROS DE USO GENERAL

Los dos grupos de seis registros de uso general B, C, D, E, H y L, y B', C', D', E', H' y L' guardan temporalmente bytes únicos de información. Debido a que

están dentro del chip del microprocesador Z-80, el intercambio de información desde uno de los registros a otro puede ser muy rápido. El intercambio de información entre cualquiera de estos registros y el acumulador es también rápido. Estos registros se pueden utilizar de uno en uno o por pares. Para el grupo 1 los tres pares de registros de 16 bits son:

- el registro de uso general de 16 bits consistente en los registros B y C. Cuando se utilizan para el direccionamiento de memoria el registro B corresponde a la posición HI de memoria y el registro C a la dirección LO.
- el registro de uso general de 16 bits consistente en los registros D y E. Cuando se utilizan para el direccionamiento de la memoria, el registro D corresponde a la dirección HI de la memoria y el registro E a la dirección LO.
- el registro de uso general de 16 bits consistente en los registros H y L que sirve para direccionar la memoria y como registro de uso general. Cuando se utiliza para direccionar la memoria, el registro H corresponde a la dirección de memoria HI y el registro L a la dirección de memoria LO.

En el grupo 2 los registros están agrupados por pares de forma similar.

ACUMULADOR

El *acumulador* es un registro de 8 bits que está en el chip del microprocesador Z-80 en el cual se coloca el resultado de muchas operaciones aritméticas y lógicas. En el caso del chip del microprocesador Z-80, el registro acumulador está situado en el mismo chip y contiene un solo byte como posibilidad de almacenamiento de memoria, es decir 8 bits. Esté atento a lo que se puede hacer con el contenido del acumulador. Por ejemplo, se pueden sumar, restar o comparar datos con el contenido del acumulador. Usted puede incrementar o decrementar su contenido de uno. Se puede intercambiar el contenido del acumulador con una posición de memoria, o con dispositivos de entrada/salida. Se pueden hacer desplazar los bits del acumulador hacia la derecha o hacia la izquierda. Se pueden realizar operaciones lógicas en el acumulador, incluyendo la Y (AND), O (OR) y O-exclusiva. Es posible que usted no pueda entender algunos de estos términos por el momento. Sea paciente, más adelante los explicaremos. Los otros registros que están en el chip del microprocesador Z-80 se discutirán con mayor detalle más adelante.

ALGUNAS INSTRUCCIONES DEL Z-80

En el capítulo 5, usted empezará a probar programas en el microcomputador.

Los programas que se ensayarán contendrán instrucciones de un solo byte, dos bytes y tres bytes, incluyendo las siguientes:

00	NOP	No operación
3C	INC A	Incrementar el contenido del acumulador de 1
76	HALT	Parar el microcomputador
3E	LD A, dato	Mover el byte de dato
<dato>		<dato>que sigue a 3E, al acumulador
32	LD (addr),A	Guardar el contenido del acumulador en la
LO		posición de memoria direccionada por los dos
HI		bytes siguientes (addr) en esta instrucción
		de 3 bytes
C3	JP addr	Salto incondicional a la dirección de memoria
LO		dada en los siguientes dos bytes en esta
HI		instrucción de tres bytes.

Por favor, observe que la lista anterior contiene instrucciones con códigos de operación que contienen un solo byte, es decir el primer byte de cada instrucción. El código hex de cada operación aparece en la primera columna cerca de su código mnemónico asociado.

Desde aquí en adelante, todos los códigos de operación, códigos de dispositivo, bytes de datos, bytes de dirección de memoria o bytes de desplazamiento se escribirán en código hex. Y todos los mnemónicos que contienen direcciones o datos llevarán los dígitos hex seguidos del carácter "H".

En las dos instrucciones siguientes se ilustra una convención de notación importante:

```
LD (direc.), A
JP  direc.
```

En la instrucción LD, "direc." está entre paréntesis, mientras que no hay paréntesis en la instrucción JP. Una dirección de 16 bits entre paréntesis representa el dato que reside en la posición direc. Por ejemplo, la instrucción

```
LD (0001H) , A
```

se ejecuta colocando el dato de un byte que está en el acumulador en la dirección especificada por 0001. El "( )" se lee como "la dirección especificada por". La dirección "direc." en la instrucción JP se refiere a la dirección de la próxima instrucción que va a ser ejecutada por el computador. Aquí, se transfiere el control del programa, mientras que en la instrucción LD se transfiere el dato. Así, una dirección de 16 bits que aparece sin paréntesis es una referencia a la misma posición, mientras que la aparición de los paréntesis implica que se hace referencia al *contenido* de aquella posición. Esta es una sutil distinción que más tarde será más natural.

NOMENCLATURA DEL BYTE DE INSTRUCCION

La literatura de Intel Corporation que describe los mnemónicos del microcomputador 8080 utiliza las siguientes abreviaciones de símbolos para el primer, segundo y tercer byte en instrucciones multibyte:

<B1> Primer byte en una instrucción
<B2> Segundo byte en una instrucción
<B3> Tercer byte en una instrucción

Extenderemos esta notación para facilitar nuestra descripción de los mnemónicos del Z-80. Por ejemplo, las instrucciones de tres bytes JP y LD se pueden escribir de la siguiente forma:

JP <B3> <B2>
LD (<B3> <B2>), A

Similarmente, las instrucciones de dos bytes LD se puede escribir como

LD A, <B2>

No-Operación: NOP

La instrucción más simple del Z-80 es la instrucción de no-operación, NOP, que tiene como código de instrucción 00.

0 0 0 0 0 0 0 0

No se realiza ninguna operación. Usted puede utilizar esta instrucción cuando desee proporcionar espacio en su programa de forma que se puedan añadir más adelante bytes de instrucción. En un capítulo posterior, usted aprenderá que su microcomputador funciona a 2,5 MHz, o 2,5 millón de estados por segundo. TODAS LAS INSTRUCCIONES DEL MICROCOMPUTADOR NECESITAN TIEMPO PARA SER EJECUTADAS. Aunque no se realice ninguna operación, es decir que no se cambia la condición de los registros y de la memoria, la instrucción NOP necesita cuatro estados, o un tiempo de 1,6 microsegundos para ejecutarse. El tiempo de ejecución depende de la velocidad del microcomputador. Si el microcomputador funcionara a 4 MHz, o 4.000.000 de estados por segundo, la instrucción NOP necesitaría un tiempo de ejecución de 1 microsegundo. Los microprocesadores Z-80A pueden funcionar a 4 MHz, con algunos chips especialmente seleccionados que funcionan algo más rápido que esto.

Discutiremos con mayor precisión lo que significa la palabra estado más adelante.

Paro: HALT

Otra instrucción simple del Z-80 es la instrucción HALT, que tiene el código de instrucción 76,

0 1 1 1 0 1 1 0

En el momento en que se ejecuta esta instrucción, el microcomputador se para. Esto se utiliza frecuentemente para permitir al microcomputador que "espere" una INTERRUPCION desde un dispositivo externo. En un computador, una *interrupción* es una rotura en el transcurso normal de una rutina de tal modo que el flujo se pueda reanudar desde aquel punto en un tiempo posterior. La instrucción HALT necesita siete estados, o un tiempo total de 2,8 microsegundos, para su ejecución.

Incrementar el Acumulador: INC A

El término *incrementar* se puede definir de la siguiente forma:

incrementar: Aumentar el valor de una palabra binaria, típicamente, aumentar el valor de 1.

La instrucción incrementar, INC A, que tiene un código de operación de 3C,

0 0 1 1 1 1 0 0

incrementa el contenido del registro acumulador de 1. La instrucción INC A necesita cinco estados, o un tiempo total de 2,0 microsegundos, para ser ejecutada.

Cargar al Acumulador el dato Inmediato: LD A, dato

Inmediato se refiere al hecho de que el byte de datos está contenido en la misma instrucción multibyte. En una instrucción multibyte se necesita un byte de datos de 8 bits o dos bytes de datos de 8 bits, que se adquieren mediante una instrucción multibyte que contiene el byte(s) de datos como byte(s) <B2> o <B3> y posiblemente <B3> o <B4>. La instrucción cargar-inmediato-al-acumulador, es una instrucción de 2 bytes que tiene como código de operación 3E, y como código mnemónico LD A, <B2>.

0 0 1 1 1 1 1 0
byte de dato

El segundo byte de la instrucción es un byte de datos de 8 bits que se deberá cargar en el registro acumulador. La instrucción completa de 2 bytes necesita siete estados, o 2,8 microsegundos, para su ejecución. Usted encontrará que ésta es una forma muy conveniente de alterar el contenido del acumulador. Es una instrucción muy popular.

Cargar el Acumulador Directo: LD (direc.), A

La instrucción cargar el acumulador directo, LD (<B3><B2>), A es una instrucción de 3 bytes cuyo código de operación es 32. Esta instrucción permite colocar el contenido del acumulador directamente en la posición de memoria, M, que está direccionada mediante los bytes segundo y tercero de la instrucción. El segundo byte es el byte de dirección baja (LO) y el tercer byte es el byte de dirección alta (HI).

```
0 0 1 1 0 0 1 0
byte de dirección LO
byte de dirección HI
```

Cuando se ejecuta la instrucción, el programa no tiene que cambiar el contenido del acumulador; simplemente copia el contenido del acumulador dentro del contenido de la posición de memoria indicada. Esta instrucción se ejecuta en 13 estados, o 5,2 microsegundos para un microcomputador que funcione a 2,5 MHz.

Salto incondicional: JP direc.

La instrucción de salto incondicional, JP <B3> <B2>, es una instrucción de 3 bytes que tiene como código de operación C3. El segundo byte de la instrucción es el byte de dirección baja (LO) de memoria, y el tercer byte es el byte de dirección de memoria alta (HI),

```
1 1 0 0 0 0 1 1
byte de direc. baja (LO)
byte de direc. alta (HI)
```

Cuando se ejecuta la instrucción, el programa salta a la dirección de memoria de 16 bits dado por los bytes de dirección HI y LO. A este tipo de instrucción se le llama una *instrucción de bifurcación.* Permite parar la ejecución secuencial del programa y saltar a otra parte de la memoria, en cuyo punto se puede continuar la ejecución del programa. Las instrucciones de bifurcación son muy potentes.

Permiten escribir *bucles* de programa, grupos de instrucciones que se ejecutan repetidamente. De esta manera, usted es capaz de reducir la complejidad del programa. La instrucción de salto incondicional necesita diez estados para ser ejecutada, o 4 microsegundos. Durante este período de tiempo se ejecuta la instrucción completa de 3 bytes.

REPASO

1. ¿Cuál es la diferencia entre un código de operación, un byte de dato, un código de dispositivo, byte de dirección y byte de desplazamiento?
2. Proporcionar el código mnemónico para los siguientes códigos de operación de 8 bits.

   a. 01110110
   b. 00111110
   c. 11000011
   d. 00110010
   e. 00111100
   f. 00000000

3. Escribir el código hex de dos dígitos para las siguientes instrucciones del Z-80.

   a. HLT
   b. JP <B2> <B3>
   c. LD (<B2> <B3>), A
   d. NOP
   e. INC A
   f. LD A, <B2>

4. En una instrucción multibyte del Z-80, el código de operación ¿puede ser el segundo, tercero o cuarto byte de la instrucción?
5. Para un microcomputador funcionando a 2,5 MHz, ¿cuánto tiempo se necesitará para ejecutar las siguientes instrucciones?

   a. JP
   b. LD A, <B2>
   c. INC A

6. Escriba los seis registros de uso general y los seis registros de uso especial en el chip del microprocesador Z-80. Indicar cuáles son los registros de uso general que se utilizan como pares de registros. ¿Qué queremos expresar mediante ARS?

RESPUESTAS

1. El código de operación es de 8 bit, 16 bit o 24 bit, para la acción específica que realizará el microprocesador Z-80. Un byte de dato es un número binario de 8 bits que utilizará el Z-80 en una operación aritmética o lógica, o para guardarlo en la memoria. Un código de dispositivo es lo que identifica al dispositivo de entrada o salida específico, con el cual el Z-80 intercambiará ocho bits de información. Un byte de dirección puede ser o bien los ocho bits más significativos o los ocho bits menos significativos en una palabra de direccionamiento de memoria de 16 bits del Z-80. Un byte de desplazamiento es un número de ocho bits en complemento a dos con signo, que se utiliza para direccionamiento indexado.

2. a. HLT
   b. LD A, <B2>
   c. JP <B2> <B3>
   d. LD (<B2> <B3>), A
   e. INC A
   f. NOP
3. a. 76
   b. C3
   c. 32
   d. 00
   e. 3C
   f. 3E
4. En una instrucción multibyte, los bytes, primero, segundo y cuarto byte pueden ser códigos de operación. El primer byte, que siempre es un byte de código de operación determina el significado del segundo byte. Si el segundo byte es un código de operación, determina al significado de los restantes byte(s), si hay alguno.
5. a. 4,0 microsegundos
   b. 2,8 microsegundos
   c. 2,0 microsegundos
6. Los seis registros de uso general son B, C, D, E, H y L.
   Los seis registros de uso especial son SP, PC, IX, IY, I y R.
   Los registros de uso general están apareados en tres registros de 16 bits de la siguiente forma: BC, DE y HL.
   Por ARS queremos expresar el grupo alternativo de registros, un segundo grupo de registros de uso general, indicadores (flags) y acumulador: A', B', C', D', E', F', H' y L'.

4

El Nanocomputador (NBZ80) y el super Nanocomputador (NBZ80S)

En el capítulo que sigue, usted realizará experimentos que demuestran conceptos importantes en la programación e interface del microcomputador. Para realizar estos experimentos usted utilizará el Nanocomputador (un microcomputador basado en el Z-80) y más tarde algunos chips de circuitos integrados, algunos zócalos extra para montar los circuitos, cable y otros componentes electrónicos. Este capítulo cubrirá alguno de estos elementos y le preparará para utilizarlos convenientemente a medida que realice los experimentos. El Nanocomputador Z-80 está fabricado por SGS-ATES COMPONENTI ELECTRONICI SpA situado en Via C. Olivetti 2-20041 Agrate Brianza-Italia.

OBJETIVOS

Al final de este capítulo, usted será capaz de hacer lo siguiente:

- Indicar la función de cada una de las 30 teclas del teclado del Nanocomputador.
- Explicar el significado de cada uno de los 14 indicadores luminosos en el teclado del Nanocomputador.
- Identificar y definir el significado de los 8 displays de siete segmentos en el teclado del Nanocomputador.
- Dar la frecuencia del reloj y el tiempo de duración de un solo estado en el Nanocomputador.

- Cargar y ejecutar un programa simple en el microcomputador.
- Indicar cuáles son los terminales que están conectados juntos en el panel para montajes sin soldadura.
- Explicar las diferencias entre memoria de lectura/escritura y la memoria programable de sólo lectura.
- Dar la posición de la memoria de lectura/escritura y de memoria sólo lectura en el Nanocomputador e indicar la dirección de inicio del sistema operativo en la memoria de sólo lectura.

EL NANOCOMPUTADOR

Propósito

El Nanocomputador de la figura 4-1 es un pequeño microcomputador basado en el Z-80 con 4K de memoria de lectura/escritura y 2K de memoria PROM/ROM. Existen dos versiones del Nanocomputador:

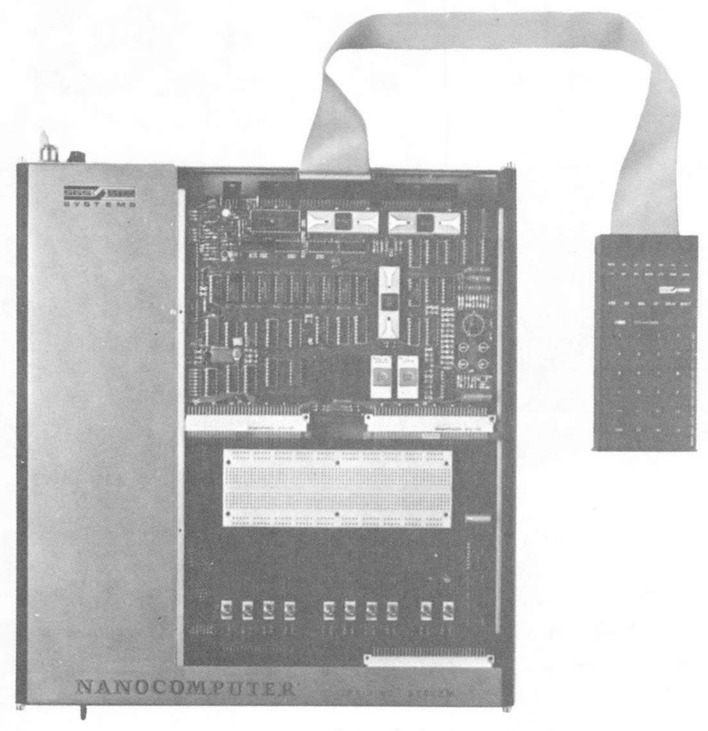
(1) El NBZ80: una tarjeta sola de microcomputador con una sección para la entrada de datos y display, a la cual nos referiremos como teclado del Nanocomputador.
(2) La tarjeta NBZ80S: una NBZ80 montada en una caja plana de sobremesa que incluye una zona para montaje de experimentos sin soldadura y una fuente de alimentación. La "S" significa super.

Los dos Nanocomputadores han sido diseñados para su utilización en la enseñanza y entrenamiento de la programación e interface de la CPU Z-80. Ambos pueden utilizarse independientemente como microcomputadores independientes, integrarse en sistemas complejos consistentes en otros microcomputadores y/o grandes computadores. Los próximos capítulos le darán experiencia en el desarrollo de software para el microprocesador y los capítulos posteriores sobre interface le introducirán a experimentos prácticos con el Nanocomputador. Usted realizará tres tipos de experimentos:

(1) Experimentos que requieren programar el Nanocomputador y que solamente utilizan el Nanocomputador NBZ80 y la fuente de alimentación NPZ80,
(2) Experimentos que incluyen la construcción de circuitos digitales y necesitan un panel de montaje (NEZ80), una fuente de alimentación y algunos componentes digitales, y
(3) Experimentos que incluyen la programación y la construcción de circuitos de interface y necesitan el Super Nanocomputador (NBZ80S).

Descripción

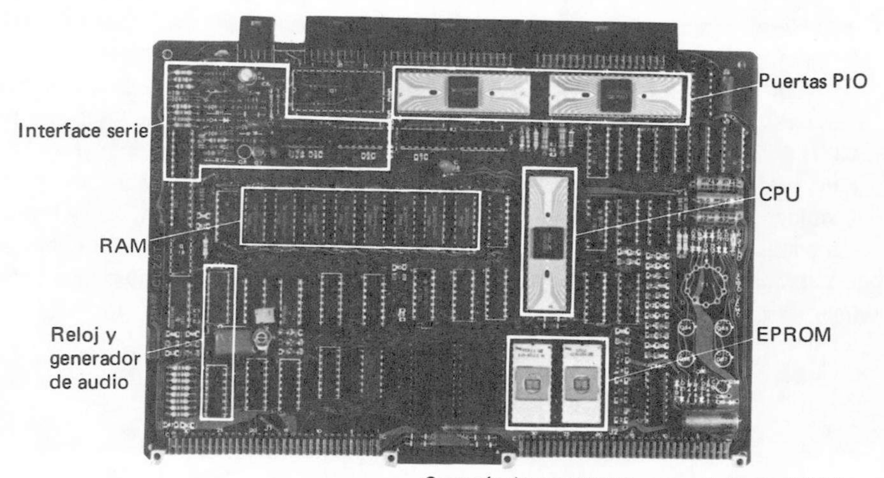
El Nanocomputador consiste en una sola tarjeta de microcomputador basado en el Z-80 con una CPU Z-80 de SGS-ATES y PIO, memoria, y una zona con un teclado de 30 teclas para entrada de datos y display. El teclado permite al usuario cargar programas en la memoria del microcomputador, seleccionar posiciones determinadas de memoria para la lectura y escritura de la memoria, ejecución de programas a toda velocidad, velocidad lenta, o un paso cada vez, reiniciar el microcomputador a un estado inicial, y muchas más funciones. Describiremos estas funciones en detalle.



Cortesía de SGS-ATES Componenti: Electronici SpA

Fig. 4-1. – El Nanocomputador NBZ80 con teclado y la fuente de alimentación NPZ80.

Varios diagramas y fotografías del Nanocomputador se muestran en las páginas siguientes. Como se muestra en las figuras 4-2 y 4-3, se pueden identificar las siguientes regiones funcionales o bloques en la tarjeta de circuito impreso del Nanocomputador:



Cortesía de SGS-ATES Componenti: Electronici SpA

Fig. 4-2.– Disposición de la tarjeta de circuito impreso del Nanocomputador.

- CPU.
- Memoria RAM.
- Memoria ROM ó EPROM.
- 4 puertas paralelas de I/O (2 circuitos integrados PIO).
- 2 puertas serie de I/O (interface serie para un terminal e interface para cassette de cinta).
- Amplificadores del Bus.
- Reloj y Generador de velocidad de transmisión.

No es necesario que usted entienda estos bloques funcionales cuando usted hace funcionar por primera vez el Nanocomputador. Inicialmente, usted estará ocupado con aprender la utilización del teclado y en leer e interpretar el display. A medida que usted desarrolle su habilidad en la programación del microcomputador, usted empezará a desarrollar una comprensión más detallada de la circuitería utilizada en el Nanocomputador.

Necesidades de alimentación

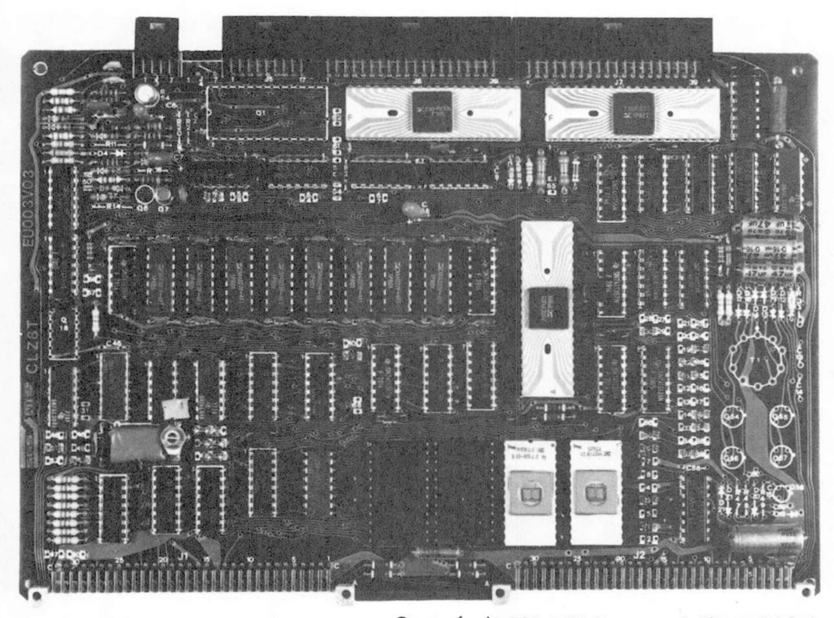
El Nanocomputador necesita una fuente de alimentación de:

+5V ±5% a 800 mA	+12V ±10% a 100 mA
−5V ±5% a 200 mA	−12V ±10% a 100 mA

Esta alimentación está incluida en la caja del NBZ80S. Para el NBZ80, SGS-ATES fabrica una fuente de alimentación adecuada (NPZ80) que se puede comprar separadamente.

Teclado del Nanocomputador

El teclado del Nanocomputador está conectado al circuito impreso mediante un cable de 40 hilos. El teclado del Nanocomputador se muestra en la figura 4-4. El siguiente texto es una descripción exhaustiva, para su referencia, de la función de cada tecla. No intente memorizar estas descripciones, si no mejor, sáltese este material en una primera lectura y espere a aprender el manejo del teclado con los experimentos al final de éste y de los capítulos siguientes.



Cortesía de SGS-ATES Componenti: Electronici SpA

Fig. 4-3. – Microcomputador en una tarjeta de circuito impreso NBZ80.

0 hasta F: Estas teclas entran un dígito hexadecimal en la posición más a la derecha del display de datos de cuatro dígitos. A medida de que los dígitos son entrados por la derecha, los tres dígitos restantes son desplazados hacia la izquierda perdiéndose el dígito que está más a la izquierda.

Flecha izquierda (←) y flecha derecha (→): Estas teclas se utilizan para seleccionar (encender) una de las 14 lamparitas que están debajo mismo de los displays de datos y direcciones. Todas las lamparitas excepto ARS, BRK y ERR se pueden encender desplazando (encender) la lámpara hacia la izquierda o hacia la derecha. Nótese que manteniendo apretada cualquiera de estas teclas provoca repetidos desplazamientos en la luz de selección. Nótese también que un desplazamiento después de la última lámpara en cualquier dirección hace que comience un nuevo ciclo.

Vamos a discutir el significado del display que se obtiene eligiendo diferentes posiciones del selector. Para las posiciones IR, AF, BC, DE y HL aparecen cuatro dígitos hex en el display de datos (los cuatro dígitos más a la derecha). Esto representa dos bytes de datos. Los dos dígitos de la izquierda muestran el contenido del registro I, A, B, D ó H, mientras que los dos dígitos de la derecha muestran el

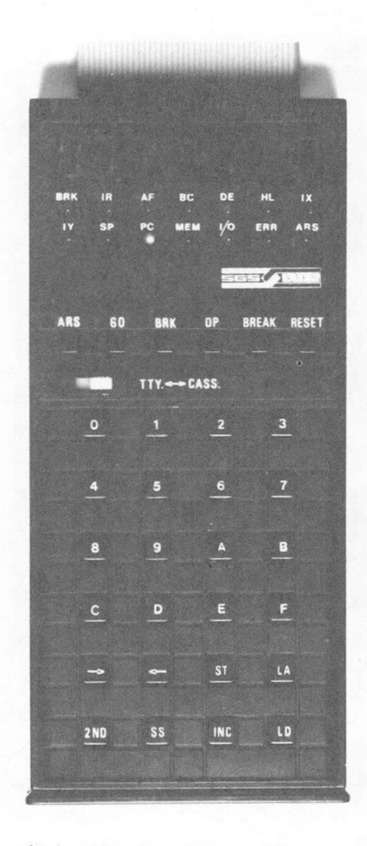


Fig. 4-4.– Teclado NBZ80.

Cortesía de SGS-ATES Componenti: Electronici SpA

contenido del registro R, F, C, E ó L dependiendo de la posición de la lámpara de selección. Para las posiciones IX, IY, SP, PC y MEM, los cuatro dígitos hex en el display de direcciones (a la izquierda) representan el contenido del registro seleccionado de 16 bits mientras que los dos dígitos hex del display de datos dan el contenido de la posición de memoria señalada por el registro de dirección.

Para la posición I/O, el display de direcciones contiene el código del dispositivo de 1 byte y el display de datos contiene el contenido del dispositivo en esta puerta.

ST: ST es una abreviación de STore (almacenar). Su función específica depende de la posición de la lámpara selectora. Si la lámpara selectora está en la posición IR, AF, BC, DE o HL, los dígitos hex que ocupan las posiciones más a la derecha (el byte de menor peso) en el display de datos son *almacenados* en el registro R, F, C, E o L, respectivamente. Si la lámpara de selección está en la posición IX, IY, SP o PC, los cuatro dígitos (dos bytes) en el display de datos son *almacenados* en los registros de 16 bits IX, IY, SP o PC.

Si la lámpara selectora MEM está encendida, los dos dígitos hex que están más a la derecha (un byte) son almacenados en la dirección que aparece en el display de direcciones y entonces el display de direcciones es incrementado (autoincrementado) para señalar a la próxima posición de memoria en secuencia.

Finalmente si la lámpara de selección I/O está encendida, los dos dígitos hex que están más a la derecha (un byte) se sacan a la puerta de salida seleccionada por el display de dirección el cual entonces se autoincrementa.

LA: LA es una abreviación de Load Address (Cargar la Dirección). Cuando la lámpara BRK está apagada [es decir, cuando el nanocomputador no está en *Modo Breakpoint* (Punto de paro)], la tecla LA se puede utilizar solamente cuando la lámpara de selección está en la posición MEM ó I/O. En cualquier otra posición del selector, si se utiliza la tecla LA, se encenderá la lámpara roja ERR para indicar que se ha intentado una operación ilegal. Cuando la lámpara BRK se enciende, la tecla LA tiene una utilización diferente. Dejaremos para más adelante la utilización en modo Breakpoint de la tecla LA hasta el párrafo que trata de la tecla BRK.

Cuando el Nanocomputador NO está en Modo BRK:

Si la lámpara de selección está en la posición MEM, LA, hace que ocurra lo siguiente:

a) Los cuatro dígitos hex que se acaban de entrar y que aparecen en el display de datos son entrados en el display de direcciones, y
b) El contenido de la posición de memoria señalado por el display de direcciones se muestra en el display de datos.

Si la lámpara de selección está en la posición I/O, LA, hace que ocurra lo siguiente:

a) El código del dispositivo de dos dígitos que se acaba de entrar y que aparece en el display de datos es entrado en el display de direcciones, y
b) El contenido de la puerta de I/O del display de direcciones se muestra en el display de datos.

2ND: 2ND se refiere al segundo byte o de mayor peso en el par de registros IR, AF, BC, DE y HL, a saber I, A, B, D y H. Para almacenar un byte (dos dígitos hex) en estos registros el procedimiento es:

PASO 1. Posicionar la lámpara de selección en el par de registros deseado (IR, AF, BC, DE, HL).
PASO 2. Entrar dos dígitos hex en el display de datos.
PASO 3. Pulsar la tecla 2ND.
PASO 4. Pulsar la tecla ST.

El resultado es que el contenido del par de registros se vuelve a mostrar en el display de datos con el byte de mayor peso cambiado de acuerdo con esto. El byte de menor peso no se cambia. Nótese que si más de dos dígitos hex se entran en el PASO 2 anterior, los dos más a la derecha (los dos últimos dígitos entrados) son los que se almacenan.

La tecla 2ND no tiene efecto cuando la lámpara de selección está posicionada en IX, IY, SP, PC, MEM o I/O, puesto que estos "registros" no son PARES de registros de 8 bits, como lo son IR, AF, BC, DE y HL.

SS: SS representa *Single Step* (paso a paso). Esto es una característica muy útil del sistema operativo del Nanocomputador en el cual los programas se pueden ejecutar en modo paso a paso. Después de cada paso, se puede examinar el contenido de varios registros, dando así al usuario una patente ayuda para depurar los programas en el proceso de desarrollo de los mismos.

También utilizaremos el modo paso a paso para ilustrar algunos de los detalles de cómo funciona el microprocesador Z-80, algo que solamente puede verse cuando el Z-80 está trabajando "a baja velocidad".

Un hecho interesante acerca de la facilidad del paso a paso en el Nanocomputador es que está implementado en software (por programa). Muchos de los sistemas paso a paso están implementados en hardware. Hablaremos más acerca de este tema fascinante, más adelante.

Se utiliza la tecla SS para cargar un programa, colocar su dirección inicial dentro del registro PC, y pulsando a continuación la tecla SS. Cada vez que se pulsa SS, se ejecuta una sola instrucción. El operador puede posicionar la lámpara de selección donde desee para elegir la información del display entre sucesivas pulsaciones de la tecla SS. La ejecución en paso a paso de un programa puede empezar después de un punto de paro (ver BRK para información sobre breakpoints).

Nótese que manteniendo la tecla SS apretada hace que el programa continúe avanzando lentamente hasta que se suelta la tecla.

INC: INC es una abreviación de *INCrementar.* Esta tecla tiene dos funciones. Primero, cuando el Nanocomputador no está en modo BRK y la lámpara selectora está en la posición MEM o I/O, provoca que la dirección de memoria o el código de dispositivo en el display de direcciones se incremente de uno, mostrando así, sucesivas posiciones de memoria o puertas de I/O y su contenido. En cualquier otra posición del selector, la utilización de la tecla INC hace que se encienda la lámpara ERR, indicando una operación ilegal.

La segunda utilización de la tecla INC se produce cuando el Nanocomputador está en el modo BRK. La utilización exacta de la tecla INC en modo BRK se describe más adelante en el párrafo acerca de la tecla BRK.

ARS: ARS significa *Alternate Register Set* (Conjunto de registros alterno). Esta tecla hace que se intercambien los dos grupos de registros A, B, C, D, E, F, H, L y A′, B′, C′, D′, E′, F′, H′, L′.

Pulsando la tecla ARS una vez, con la lámpara de selección en la posición AF, usted hace que se muestren en el display de datos los registros A′ y F′. Los resultados son similares para las posiciones BC, DE y HL de la lámpara selectora.

Cuando se muestra al conjunto de registros alternos. La lámpara ARS se enciende. Obsérvese que pulsando la tecla ARS varias veces, la lámpara ARS se enciende y se apaga.

GO: Esta tecla tiene dos funciones. La primera función es la de iniciar o continuar la ejecución de un programa del microcomputador. La otra función es la de eliminar puntos de paro, lo cual se discutirá en el párrafo que trata de la tecla BRK. Para iniciar la ejecución de un programa en el microcomputador, se debe especificar la dirección inicial del programa. Esto puede hacerse de cualquiera de estas dos formas:

1. Cargar el PC (Contador de Programa) con la dirección inicial, pulsando entonces GO para empezar la ejecución.
2. Entrar la dirección de inicio en el display de datos pulsando inmediatamente GO.

En ambos casos, se realizará la ejecución del programa hasta que, el programa se pare, o se devuelva el control al sistema operativo, o se encuentre un punto de paro (BRK). Para continuar la ejecución después de un punto de paro, simplemente pulse de nuevo GO.

BRK: BRK es una abreviación de *breakpoint* (punto de paro). Esta tecla es un interruptor que coloca al Nanocomputador en el modo BRK o fuera de él. Cuando está en el modo Breakpoint, la lámpara BRK se enciende; de lo contrario está

apagada. Así, pulsando alternativamente la tecla BRK se hace que la lámpara se encienda y se apague.

Vamos a discutir en que consiste un punto de paro y que significa para el Nanocomputador estar en el Modo BRK. Un punto de paro provoca un alto en la ejecución del programa. Usted define un punto de paro, especificando una instrucción del programa en donde debe detenerse la ejecución. Usted puede entonces examinar los registros y la memoria antes de continuar la ejecución del programa. La ejecución se puede continuar en modo paso a paso (utilizando la tecla SS), o a toda velocidad (pulsando de nuevo GO). Usted especifica un punto de paro dando una dirección (dos bytes o cuatro dígitos hex). Esta dirección DEBE señalar al primer byte de la instrucción multibyte. Es importante recordar que la instrucción que empieza en la dirección del punto de paro NO es ejecutada cuando se encuentra el punto de paro. Esta instrucción se ejecutará solamente cuando continúe la ejecución del programa.

Usted puede definir hasta ocho direcciones con puntos de paro de una vez. Estas direcciones están numeradas del 0 al 7. Esta numeración es solamente para facilidad de referencia y no significa nada acerca del orden en el que deben introducirse los puntos de paro. Por ejemplo, el punto de paro 0 puede suceder más tarde en la ejecución del programa que el punto de paro 5. La siguiente secuencia de pasos describe como se definen los puntos de paro así como las funciones particulares de las teclas BRK, INC y LA en el proceso de la ejecución del programa.

Paso 1: Pulsar la tecla BRK para entrar en el modo BRK. El Nanocomputador está ahora preparado para aceptar definiciones de puntos de paro.

Paso 2: En el display de datos debe aparecer un solo dígito. Pulsando la tecla INC hace que este número sea incrementado de uno hasta el número 7 y entonces de nuevo a cero. Este dígito indica el punto de paro actual.

Paso 3: Definir la dirección del punto de paro para el número deseado de punto de paro, mostrando primero un solo dígito que es el que se desea, y pulsando a continuación una dirección de cuatro dígitos seguida de la tecla LA. El display resultante debe ser tal que:

a) Los primeros cuatro dígitos son la dirección que usted ha entrado.
b) Hay un espacio en blanco seguido de tres dígitos.
c) El primer dígito del grupo de tres números es el del punto de paro.
d) El segundo y tercer dígitos del grupo de tres dígitos son el contenido de la dirección del punto de paro, es decir el primer byte de la instrucción del punto de paro.

Paso 4: Se pueden definir puntos de paro sucesivos por medio de INCrementar hasta el punto de paro deseado, entrando una dirección, y pulsando LA.

Cualquier punto de paro que ya ha sido definido se puede cambiar utilizando el mismo procedimiento.

Paso 5: Pulsar la tecla BRK de nuevo para salir del Modo Breakpoint.

Para eliminar un punto de paro, entrar en el modo BRK, INCrementar hasta el punto de paro que se desea eliminar, y pulsar GO. El display resultante debe contener solamente el número del punto de paro.

LD y DP: Si su Nanocomputador está equipado con un interface para cinta de cassette. Usted puede utilizar los cassettes como medio de almacenamiento de gran capacidad. Esto es, usted puede volcar a la cinta los programas que están guardados en la memoria (la tecla DP); o usted puede cargar desde la cinta los programas o datos que se han guardado previamente (la tecla LD).

Puesto que el contenido de la memoria de lectura/escritura se destruye siempre que se apaga la alimentación del Nanocomputador, un medio de almacenamiento de gran capacidad como una cinta de cassette facilita mucho el devolver la memoria a un estado deseable después de conectar la alimentación. De hecho, cuando usted realice los experimentos de programación en éste y en los capítulos siguientes, le recomendamos encarecidamente que usted vuelque los programas largos a cinta, después de haberlos introducido a mano. De esta forma si es necesario volver a cargar el programa, usted no tendrá que volver a teclear cada byte . . . ¡solamente pulsar la tecla LD!

Es importante mencionar que las teclas LD y DP se pueden utilizar con otros dispositivos que los registradores de cinta magnética. Cualquier dispositivo serie ASCII puede enviar o recibir datos a/desde el Nanocomputador. Esto es, los mandos LD y DP se pueden utilizar para entrada/salida de datos a cintas de papel, perforadoras de cinta, impresoras CRT, etc. Usted solamente informa al Nanocomputador, mediante el interruptor TTY/CASS en el teclado, cuando es necesario un teletipo digital serie (TTY) o un registrador cassette de audio (CASS). Discutiremos específicamente la entrada/salida I/O al cassette de audio, con más detalle, más adelante.

Vamos ahora a discutir como utilizar el grabador de cassette RCZ80. Entonces describiremos completamente las operaciones de carga y volcado.

El registrador de audio del tipo cassette RCZ80 es proporcionado por SGS-ATES para ser conectado con el Nanocomputador. Sin embargo, se puede utilizar cualquier grabador cassette estándar. Esta discusión se aplica específicamente a la unidad RCZ80, aunque con solamente ligeras modificaciones puede ser aplicada a equipos fabricados por otros. Para estos nos referiremos al manual de funcionamiento del equipo en cuestión.

Para disponer el grabador cassette para funcionamiento:

- seleccionar la tensión de alimentación apropiada (110/120 ó 220/240 V de alterna) utilizando el interruptor situado en la parte posterior del registrador;

- conectar la tensión alterna;
- hacer girar el control de volumen al máximo (10);
- con la tensión apagada, conectar el cable del cassette: El conector redondo de siete patillas se enchufa en el zócalo situado al lado del registrador, y el conector plano de ocho patillas dentro del conector J3 de la tarjeta NBZ80 (parte superior izquierda);
- posicione el interruptor TTY/CASS en el teclado del Nanocomputador en CASS.

El funcionamiento DP

La tecla DP se utiliza para iniciar la operación de escribir al cassette. El procedimiento para grabar el contenido de un bloque contiguo de memoria del Nanocomputador (RAM, ROM o EPROM) es el siguiente:

1. Aplicar tensión al Nanocomputador.
2. Posicionar la cinta a la posición de grabación inicial (utilizar las teclas FORWARD (>>) y REWIND (<<).
3. Posicione la lámpara selectora en el teclado del Nanocomputador en MEM.
4. En el teclado del Nanocomputador, entre la primera dirección del bloque de memoria que usted desea volcar (hasta cuatro dígitos hex) y pulse entonces la tecla LA.
5. En el teclado del Nanocomputador, entrar la longitud del bloque de memoria que usted desea volcar (hasta cuatro dígitos hex). El display del teclado debe mostrar ahora la dirección de inicio a la izquierda y la longitud del bloque a la derecha.
6. Pulsar la tecla DP en el teclado del Nanocomputador. El display del teclado quedará apagado. Ha empezado la operación de escritura en el cassette.
7. Confirmar que el interruptor TTY/CASS está en la posición CASS. Apretar simultáneamente la tecla roja RECORD y la tecla FORWARD (>). La cinta no empezará a moverse por ahora.
8. Pulsar la tecla GO. La cinta empezará a moverse lentamente. Después de unos 20 segundos de un tono continuo (que se escribe como una cabecera), el Nanocomputador empezará a grabar datos en la cinta de cassette.
9. Cuando termina la operación de escritura al cassette, éste se parará automáticamente. Nuestra experiencia es que para escribir 256 bytes se necesitan 20-25 segundos, así que no se alarme si la operación de escritura parece durar más tiempo del que usted esperaba. Después de que el cassette se para, pulsar cualquiera de las teclas > > ó < < para reconocer el final de la operación de escritura.

10. Pulsar cualquier tecla en el teclado del Nanocomputador para volver al funcionamiento normal. También en este momento, usted puede rebobinar la cinta y quitarla del registrador, si así lo desea.

Funcionamiento LD

La tecla LD se utiliza para cargar programas y/o datos grabados previamente en un cassette a la memoria. La dirección de inicio y el número de bytes ha sido guardado en la cinta al mismo tiempo que el programa y los bytes de datos, de forma que no es necesario especificarlos de nuevo. Aquí está la secuencia de manipulaciones para una operación de lectura de cassette:

1. Aplicar la alimentación al Nanocomputador.
2. Posicionar la cinta a la posición inicial de lectura (utilice las teclas FAST FORWARD (> >) y REWIND (< <).
3. Vuelva a comprobar que el interruptor TTY/CASS en el teclado del Nanocomputador está en la posición CASS.
4. Pulsar la tecla LD. El display del Nanocomputador quedará apagado. Se ha iniciado la operación de carga.
5. Pulse la tecla FORWARD (>) y escuche el sonido en tono alto de los datos.
6. El cassette se parará cuando esté terminada la operación de lectura del cassette. Usted también oirá un sonido inconfundible que no intentaremos describir aquí.
7. Cuando el cassette se ha parado, pulse la tecla FAST FORWARD (> >) o REWIND (< <) para reconocer el final de la operación de lectura.
8. Si la lamparita de ERR en el Nanocomputador está encendida, se ha producido un error de "checksum" (error en la transmisión de datos). Intente leer la cinta de nuevo. Si la lámpara ERR se enciende de nuevo, el cassette es defectuoso o está escrito incorrectamente.
9. Si la carga ha tenido éxito, la lámpara ERR estará apagada.
10. Pulse cualquier tecla en el Nanocomputador para restablecer el funcionamiento normal.

NOTA 1:

Al terminar cualquiera de las operaciones de carga o volcado, el movimiento del cassette queda bloqueado hasta que se recibe un reconocimiento (pulsando las teclas de bobinado rápido o de rebobinar en el registrador) y se recupera el funcionamiento normal pulsando cualquier tecla en el teclado del Nanocomputador. Esta secuencia final es muy importante, de forma que siempre debe seguirla exactamente.

NOTA 2:

La velocidad de transmisión serie en I/O durante la lectura y escritura de cassettes es programable. Cuando el Nanocomputador es alimentado o se pulsa RESET, la velocidad se coloca a 600 baud o 60 caracteres por segundo, en donde cada carácter consta de 10 bits (2 bits de inicio, 7 bits de código ASCII, y 1 bit de paro). Cambiando el contenido de las posiciones de memoria BAUDRT y BAUDRT +1 se pueden obtener velocidades serie de I/O de 110 y 300 baud. La siguiente tabla da la correspondencia entre el contenido de las posiciones BAUDRT y BAUDRT +1 y las velocidades serie de I/O.

(BAUDRT)	(BAUDRT + 1)	Velocidad en BAUD
9A	00	600
35	01	300
55	03	110

La dirección absoluta asociada con la etiqueta BAUDRT puede ser obtenida de la Tabla Maestra de Símbolos en el apéndice F.

NOTA 3:

El formato de los bytes almacenados en una cinta es el siguiente:

- Cada byte de memoria es trasladado a dos caracteres ASCII, un carácter ASCII para cada "hex nibble" (medio byte). Por ejemplo, el byte de memoria binario 00101010 es trasladado a ASCII 2 y un ASCII A.
- Los bytes de memoria se agrupan en ocho cada registro.
- Cada registro tiene el formato:

Caracteres	Contenido
1	Retorno de carro
2	Avance de línea
3	Dos puntos
4–5	Longitud del registro
6–7	Dirección de memoria, de inicio de la grabación, byte HI
8–9	El registro empieza en la dirección de memoria, de inicio de la grabación, byte LO
10–11	No utilizados
12–N	Byte de memoria de datos, bytes de 2 caracteres ASCII por byte (el número depende de la longitud del registro especificado en los caracteres 4-5)
(N+1)–(N+2)	Suma de comprobación

NOTA 4:

El Nanocomputador es capaz de leer una cinta de cassette grabada por otro producto SGS-ATES llamado el CLZ80 que es un microcomputador basado en el Z-80, si la cinta fue creada utilizando el Monitor/Depurador o el software ensamblador ASS-Z. Esto se realiza cargando la posición de memoria INMODE (ver la

Tabla Maestra de Símbolos en el apéndice F para la dirección absoluta) con cualquier byte que no sea igual a 00 (hex), y siguiendo entonces el procedimiento usual de carga del Nanocomputador. Nótese que las cintas creadas por el Nanocomputador no se pueden leer mediante un CLZ80 funcionando con el Monitor/Depurador MO-Z o el software del Ensamblador ASS-Z.

BREAK: Puede pensarse que la tecla BREAK es un "botón de pánico". Pulsando la tecla BREAK se provoca una interrupción no enmascarable de la CPU (NMI) la cual, a su vez provoca una detención inmediata del programa que está ejecutando. Se devuelve el control al sistema operativo del Nanocomputador con la lámpara de selección colocada en la posición PC. El display de direcciones señala a la última instrucción ejecutada y los otros registros se conservan como estaban después de que se ejecutara la última instrucción del programa.

RESET: La función de la tecla RESET es la de restaurar el Nanocomputador a su estado inicial. Al principio de la ejecución del sistema operativo se efectúa una inicialización. Así todos los registros son colocados a cero y se borran todas las direcciones de los puntos de paro preexistentes.

Interruptor CASS/TTY: Para los Nanocomputadores que disponen de un interface para registradores de cassette y para un terminal serie del tipo teletipo, este interruptor selecciona uno de los dos dispositivos para la transmisión serie de I/O.

Esto termina nuestra discusión acerca del teclado del Nanocomputador basado en el Z-80.

UNIDAD CENTRAL DE PROCESO (CPU)

El Nanocomputador es un sistema de microcomputador basado en un Z-80. El chip Z-80 de 40 patillas (DIP) fue originalmente diseñado y producido por Zilog Corporation en 1976. El chip del microprocesador Z-80 se fabrica actualmente en Europa en la fábrica de SGS-ATES en Italia.

Reloj

El cristal de cuarzo se encuentra en la esquina inferior izquierda de la tarjeta del Nanocomputador y tiene una frecuencia de 2,4576 MHz. Asociado con este cristal está un chip generador de reloj y driver que saca una frecuencia de reloj de 2,4576 MHz. Esta frecuencia de reloj de 2,4576 MHz manda el chip del microcomputador Z-80, en cada uno de los pasos de cálculo que realiza. La frecuencia máxima que se puede aplicar al Z-80A es de 4 MHz. Desafortunadamente a esta

frecuencia, los dispositivos PROM tales como las EPROM 2708 o la 2716 no son bastante rápidas. Es por esta razón que SGS-ATES ha elegido la CPU Z-80 estándar que funciona aproximadamente a 2,5 MHz. Obsérvese que a 2,5 MHz un solo estado, o ciclo de reloj, tiene una duración de 400 nanosegundos, o de 0,4 microsegundos.

Memoria

La memoria del Nanocomputador está compuesta de memoria dinámica de lectura/escritura y de memoria de sólo lectura. La memoria RAM dinámica disponible en el Nanocomputador es de 4K bytes pero puede ser expandida a 16K bytes. La memoria de lectura/escritura se direcciona en la región de los primeros 4K: posiciones 0000H hasta 0FFFH. La ROM en el Nanocomputador es de 2K bytes y se puede expansionar hasta 8K bytes. La ROM se direcciona en la zona de 2K a 8K, dependiendo del tamaño de la ROM.

Puertas de I/O

El Nanocomputador está provisto de puertas de I/O en serie y en paralelo. Dos chips Z80-PIO implementan I/O paralelo, mientras que un dispositivo serie y un circuito interface para registrador de audio implementan I/O serie. Mientras que un chip PIO se utiliza para el interface del display y del teclado, el otro chip PIO está disponible para su utilización por el usuario. Los circuitos serie se pueden utilizar para conectar el Nanocomputador con muchos terminales serie a una velocidad de transmisión de 110 baud; y el circuito de interface digital de audio está para mandar un registrador de cassette.

Descripción del tablero

Para muchos experimentos que usted realizará con su Nanocomputador se le pedirá que construya circuitos eléctricos utilizando un tablero, cables, chips de circuitos integrados, y otros componentes eléctricos. Aquí daremos una breve descripción del tablero; sin embargo, cubriremos este tema más a fondo más adelante.

El tablero está diseñado para acomodar los muchos experimentos que usted realizará en los siguientes libros. Los chips de circuitos integrados, resistencias, condensadores, cables y los dispositivos lógicos adicionales se conectan todos o se introducen directamente en el tablero.

En las figuras 4-5 y 4-6 se muestran vistas por encima y por debajo del tablero. El tablero contiene 128 conjuntos de 5 terminales para no soldar conectados eléctricamente y 2 conjuntos de 64 que cubren ambos lados. Además hay 8 grupos de 25 contactos para no soldar, conectados eléctricamente a lo largo de los bordes del tablero. El término para no soldar se utiliza aquí porque usted realiza conexiones eléctricas entre los componentes electrónicos sin necesidad de soldar y de un soldador.

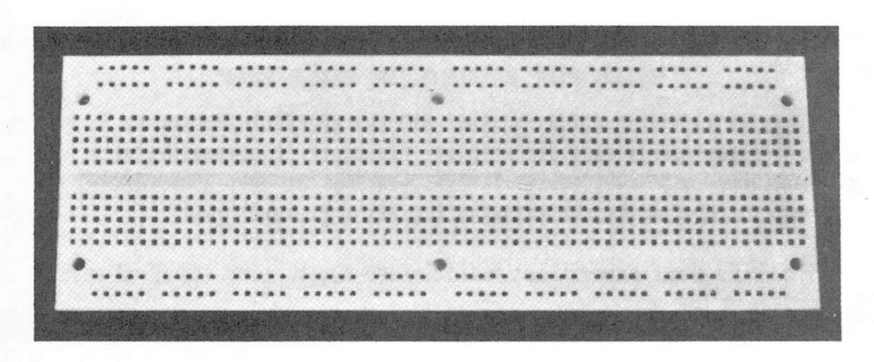


Fig. 4-5.– Vista superior del tablero para montajes sin soldadura.

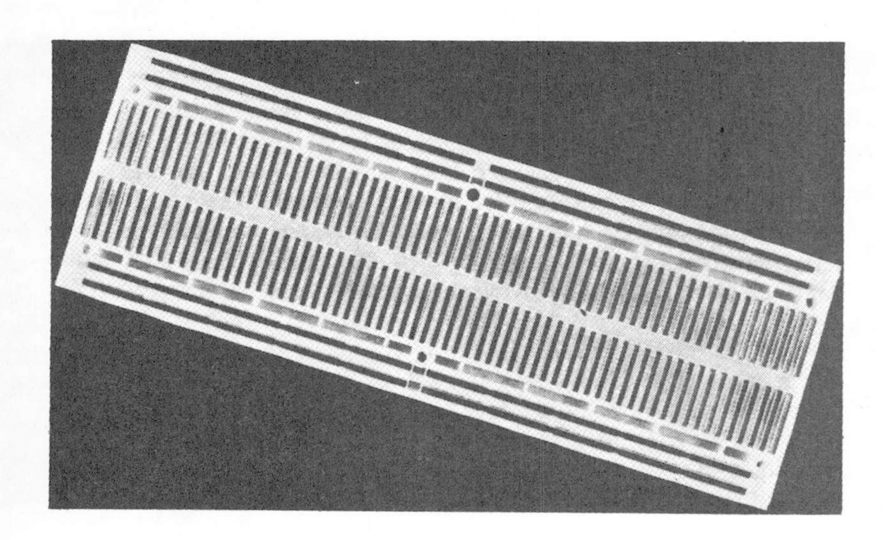


Fig. 4-6.– Vista inferior del tablero para montajes sin soldadura.

El grupo central de 5 terminales conectados eléctricamente acomodan los chips de circuitos integrados y permiten que se puedan realizar cuatro conexiones adicionales a cada patilla para los chips de circuitos integrados más pequeños de 14 o 16 patillas. Los grupos de 25 contactos conectados eléctricamente en los extremos del tablero son conectados a +5 V o a masa. Son los que proporcionan la alimentación a los chips de circuitos integrados y para otras funciones del tablero de montaje, que se describirán en otro capítulo.

REGLAS PARA MONTAR LOS EXPERIMENTOS

En los siguientes capítulos, usted utilizará el Nanocomputador para realizar experimentos que demuestran conceptos de la programación e interface del microcomputador. Antes de que ponga a punto un experimento, le recomendamos que usted observe las siguientes reglas generales:

1. Planee sus experimentos de antemano. Conozca el tipo de resultados que usted espera observar.
2. Limpie el tablero de montaje de todos los cables innecesarios y componentes de los experimentos previos.
3. IMPORTANTE: Antes de realizar cualquier montaje, desconecte la conexión del cable de +5 V del bus exterior en el zócalo de montajes. Observe que no le hemos pedido que desconecte la alimentación de todo el microcomputador porque al hacer esto, usted borrará toda la memoria de lectura/escritura.
4. Con la tensión de +5 V desconectada del tablero de montaje, cablee cuidadosamente el circuito de interface al microcomputador. Conecte los cables de alimentación a los circuitos integrados antes de realizar cualquier otra conexión.
5. Ponga mucha atención en la posición de los varios chips en el zócalo de cableado. La situación juiciosa de estos dispositivos puede minimizar frecuentemente la jungla de cables de conexión que es inherente a cualquier circuito digital de complejidad modesta.
6. Compruebe el cableado del circuito para asegurarse de que es correcto. PONGA ESPECIAL ATENCION A LAS CONEXIONES DE ALIMENTACION DE LOS CHIPS DE LOS CIRCUITOS INTEGRADOS. Si están equivocados, usted destruirá su chip y tal vez borre la memoria de lectura/escritura. Utilice el dedo colocándolo en la superficie superior del chip para determinar si está muy caliente; si está caliente usted ha hecho algo equivocado.
7. Aplique los 5 volt de tensión de alimentación cuando todo ha sido comprobado. Usted puede entonces aplicar el test de "tacto" para determinar si alguno de los chips se calienta excesivamente.

8. Una vez que usted ha terminado con los experimentos, no desconecte el circuito. Antes de hacerlo, mire el siguiente experimento para determinar si este emplea o no el mismo circuito.
9. Desconecte la alimentación principal del microcomputador cuando usted ha terminado la jornada de trabajo. Si usted tiene un interface de cassette en su Nanocomputador, usted puede almacenar sus programas antes de desconectar la alimentación, una práctica que es muy recomendable, puesto que al desconectar la alimentación se borra la memoria de lectura/escritura.

FORMATO PARA LAS INSTRUCCIONES DE LOS EXPERIMENTOS

Las instrucciones para cada experimento se presentan en el siguiente formato.

Propósito

El material que se presenta bajo esta cabecera especifica el propósito del experimento. Será útil para usted el tener en la mente este enunciado mientras usted realiza el experimento.

Configuraciones de las patillas de los chips de los circuitos integrados

La configuración de las patillas, dada con el permiso de SGS-ATES, se da bajo esta cabecera para todos los chips de circuito integrado utilizados en el experimento. Nótese que todos los experimentos utilizan chips de SGS-ATES TTL Schottky de baja potencia. Si el circuito es idéntico al dado en el experimento inmediatamente anterior, la configuración de las patillas puede ser omitida.

Diagrama esquemático del circuito

Usted recibirá el diagrama esquemático del circuito completo que usted cableará en el experimento. Usted debe analizar este diagrama en un esfuerzo para entender el circuito antes de que usted continúe con la experimentación. Compruebe todos los números de las patillas de todas las conexiones a los chips de circuitos integrados. TENGA PRESENTE QUE LAS CONEXIONES DE ALIMENTACION DE LAS PUERTAS HA SIDO OMITIDA. Ponga especial atención a las conexiones del +5 V y masa; su circuito cableado no funcionará si se omite cualquiera de ellas.

Programa

Se le proporcionará el programa del microcomputador que usted debe cargar en las posiciones de memoria indicadas.

Pasos

Bajo el título de cada paso en secuencia, por ejemplo Paso 1, Paso 2, etc., están las instrucciones detalladas concernientes a cómo debe realizar esta parte del experimento. Usted debe responder a las preguntas en el momento en que está realizando el experimento. Después de que haya escrito su respuesta, determinar si la respuesta correcta está indicada en el texto que sigue inmediatamente a la pregunta. Si éste es el caso, o si las dos respuestas no coinciden, asegúrese de que usted entiende la discrepancia (y la respuesta correcta, si es posible) antes de que usted continúe más adelante con el experimento.

Preguntas

A menudo se harán preguntas que probarán: (a) su comprensión del experimento que usted acaba de terminar, (b) su habilidad para anticiparse a experimentos futuros o problemas, (c) su habilidad para relacionar material en forma de texto, con la información determinada experimentalmente, y utilizando esta información, formular respuestas a las preguntas que cubren el material al cual usted no ha sido introducido previamente. El número de preguntas proporcionado dependerá de la naturaleza del experimento, lo avanzado que esté en el libro, la fase de la luna y de la fatiga de los autores. En muchos capítulos las preguntas se consolidarán con una sección de Repaso al final del capítulo. Se proporcionarán respuestas para cada sección de Repaso.

UNA PALABRA DE PRUDENCIA

Para los programadores novatos del microcomputador que están utilizando este texto, deseamos dejar perfectamente claro un punto:

ES IMPOSIBLE ESTROPEAR UN MICROCOMPUTADOR POR UNA PROGRAMACION IMPROPIA

Usted puede borrar el contenido de la memoria de lectura/escritura, pero usted

no puede destruir o estropear el sistema del microcomputador si usted hace errores en un programa e intenta entonces ejecutarlo. Así que, relájese y diviértase con su microcomputador. Haga equivocaciones en la programación. Aprenda de ellas.

Usted puede estropear su sistema de microcomputador si usted:

- Le aplica la alimentación incorrectamente.
- Permite que materiales metálicos cortocircuiten accidentalmente cualquier interconexión de cables en la tarjeta de circuito impreso.
- Realiza conexiones equivocadas con un cable a un interface. Ponga especial atención a la entrada de datos en el bus de datos; ENTRE TODOS LOS DATOS CON LA AYUDA DE BUFFERS DE TRES ESTADOS.
- Lo deja caer.
- Le hace funcionar en un ambiente excesivamente caliente o corrosivo.
- Tratar de repararlo sin saber lo que se está haciendo.

Muchos instrumentos de laboratorio están alojados en una caja metálica o de plástico y ofrecen, hasta un cierto punto, protección para la electrónica contra un usuario poco cuidadoso. El Nanocomputador (NBZ80 está a la vista, de forma que usted pueda observar cómo está construido y cómo funciona; sin embargo, es vulnerable, como resultado de esta exposición. Pensamos que es importante que usted no esté intimidado por su microcomputador, y que no esté oculto para usted con un chasis opaco.

Recuerde que si usted solamente está programando su microcomputador, usted no puede estropearlo. Si usted está programando y haciendo interfaces con su microcomputador, debe de tener cuidado. En ciertos casos usted puede estropear el chip del Z-80 mediante una mala programación si usted utiliza un circuito de interface incorrectamente.

Le pedimos que sea cuidadoso.

INTRODUCCION A LOS EXPERIMENTOS

Los siguientes experimentos están diseñados para demostrar el funcionamiento de las varias teclas del teclado del Nanocomputador. Para llevar a cabo estos experimentos usted necesitará:

1 Nanocomputador (NBZ80 ó NBZ80S) que esté en un buen estado de funcionamiento,

1 teclado para el Nanocomputador y su software PROM/ROM (suministrado con el Nanocomputador),

1 fuente de alimentación para su NBZ-80 (el NBZ80S tiene su propia fuente).

Los experimentos que usted realizará se pueden resumir de la siguiente forma:

Experimento N.°	Comentarios
1	Demuestra el funcionamiento de las teclas numéricas del 0 hasta la F, la tecla LEFT-ARROW (flecha izquierda), y de la tecla RIGHT-ARROW (flecha derecha) en el teclado del Nanocomputador.
2	Demostrar la función de las teclas ST y 2ND para cargar los registros con datos.
3	Demuestra la carga de información presente en el DISPLAY DE DATOS dentro de una posición específica de memoria en la memoria de lectura/escritura. También se demuestra la función de la tecla INC como un medio excelente de mostrar el contenido de posiciones sucesivas de memoria.
4	Demuestra la carga y ejecución de programas muy simples del microcomputador en modo paso a paso (tecla SS). También demuestra la función de las teclas GO y RESET.
5	Demuestra dos rutinas de utilidad del Nanocomputador que residen en memoria de sólo lectura: el test de la RAM y el test del Teclado/Display.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de comprobar el funcionamiento de las teclas numéricas, la tecla FLECHA IZQUIERDA y de la tecla FLECHA DERECHA en el teclado del Nanocomputador.

Paso 1

Aplique tensión al Nanocomputador y pulse la tecla RESET. Usted observará

cuatro dígitos hex en el display de direcciones, dos dígitos hex en el display de datos, y la lámpara selectora debe estar en la posición PC. En nuestro caso la dirección del display indica 0000 y en el display de datos indica 00.

Paso 2

Pulse la tecla 0. ¿Qué se observa en el registro de datos?

Observamos un 0 a la derecha precedido de tres espacios en blanco en el registro de datos. Si esto no es lo que ha sucedido, entonces es que usted tiene un problema con su Nanocomputador. Debe comprobarlo antes de continuar. (Véase lo que está escrito en el Experimento N.° 6 de este capítulo acerca de los test de autodiagnóstico de la memoria y teclado/display.)

Paso 3

Pulsar la tecla 1 y a continuación la 2. El 0 debe haber sido desplazado hacia la izquierda de dos dígitos para dejar espacio para el 1 y 2. Así el display de datos debe contener un espacio en blanco seguido por 012. Entre un 3, y después un 4. ¿Qué es lo que usted observa ahora?

Observamos que el 0 ha salido del display para hacer espacio para el dígito 4. El display de datos tiene cuatro dígitos iluminados: 1234.

Paso 4

Continúe entrando dígitos. Cada vez que se entra un dígito, el display de datos de cuatro dígitos se desplaza hacia la izquierda desapareciendo el dígito que está más a la izquierda. ¿Qué le ha sucedido al display de direcciones? ¿a la lámpara selectora?

El display de dirección y la lámpara de selección permanecen igual.

Paso 5

Pulse ahora la tecla FLECHA DERECHA (→) una vez. ¿Qué es lo que usted observa?

Observamos varios cambios. Primero el display de datos y direcciones cambian para leer 0000 y 00 respectivamente. Su Nanocomputador puede que no haya producido exactamente estas lecturas, pero debe de haber ocurrido un cambio. También la lámpara selectora se ha movido ahora a la posición MEM. Usted también debe haber observado esto. El significado de este display es que la posición de memoria 0000 contiene el byte de dato 00.

Paso 6

Pulse la tecla FLECHA IZQUIERDA (←). Su display primitivo debe ser restaurado. El nuestro era 0000 00. Esto significa que el registro PC de 16 bits contiene 0000 y que el contenido de la posición de memoria 0000 es 00.

Paso 7

¿Cómo puede usted posicionar la lámpara selectora en AF?

Existen dos formas de hacer esto, una forma es utilizando la tecla FLECHA IZQUIERDA (←), y otra forma mediante la tecla FLECHA DERECHA (→). Con cualquiera de las dos teclas, simplemente manténgala en la posición pulsada hasta que se alcance la posición deseada. La lámpara se enciende de un paso a la vez, iluminando cada una de las lámparas de selección hasta que se alcance AF.

Paso 8

¿Qué es lo que usted observa en los displays de direcciones y datos con el selector en la posición AF?

Observamos un display de direcciones apagado (usted también) y un display de datos con los cuatro dígitos hex 0000. Esto representa el contenido del par de registros AF, es decir el acumulador contiene 00 y los indicadores de estado contienen 00. Su display de datos debe contener cuatro dígitos hex.

Paso 9

Continúe eligiendo diferentes posiciones del selector. ¿Qué es lo que puede usted decir acerca del display de direcciones y de datos para las posiciones IR, AF,

BC, DE, HL? ¿Qué, acerca de las posiciones IX, IY, SP, PC y MEM? ¿Qué, acerca de la posición I/O?

Observamos que las posiciones IR, AF, BC, DE y HL producen un display de direcciones apagado y cuatro dígitos del display de datos que significa el contenido del par de registros seleccionado. Las posiciones IX, IY, SP, PC y MEM dan un display de cuatro dígitos de direcciones y un display de datos de dos dígitos. Para las posiciones IX, IY, SP y PC, el display de direcciones da el contenido del registro de 16 bits seleccionado, mientras que el display de datos da el contenido de la posición de memoria direccionado por el registro. El significado del display para la posición MEM se da en el Paso 5.

En la posición I/O aparecen dos dígitos justificados a la derecha en los displays de direcciones y datos. El display de direcciones contiene el código del dispositivo, mientras que el display de datos representa el byte de información que está en este momento en aquel dispositivo. Pasaremos bastante tiempo describiendo las I/O y los códigos del dispositivo en capítulos posteriores.

EXPERIMENTO N.º 2

Propósito

El propósito de este experimento es el de demostrar la función de las teclas ST y 2ND para cargar los registros con datos.

Paso 1

Mover la lámpara selectora a la posición BC, entre los dígitos hex 11 dentro del display de datos, y pulse la tecla ST. ¿Qué es lo que usted observa?

Usted debe ver que el display de datos, que representa el contenido del par de registros BC, tiene 11 como dígitos situados más a la derecha. En otras palabras usted ha guardado un byte (11 hex o 00010001 en binario) en el registro C de la CPU del Z-80.

Paso 2

Pruebe ahora a entrar 22 en el display de datos, pulsando a continuación la tecla

2ND. ¿Qué se lee en el display de datos ahora? ¿Qué registro del Z-80 ha cambiado esta vez?

Debe leer 2211. Usted acaba de guardar un byte en el registro de mayor peso, es decir el registro B, del par de registros BC.

Paso 3

¿Cómo guardaría usted los bytes 2103 en el par de registros HL?

RESPUESTA: Pulse las siguientes teclas en secuencia.

1. FLECHA DERECHA (→), hasta que la lámpara de selección esté en la posición HL
2. 2
3. 1
4. 2ND
5. ST, para almacenar el byte de mayor peso 21 en H
6. 0
7. 3
8. ST, para almacenar el byte de menor peso 03 en L

¿Existe alguna otra forma de hacer lo mismo?

La respuesta es sí; la tecla utilizada para posicionar la lámpara de selección podría haber sido la tecla FLECHA IZQUIERDA (←) y se habría guardado el byte de menor peso (03) en el registro L antes de que se hubiera guardado el byte de mayor peso (21) en el registro H. Así con el Nanocomputador y sus flexibles operaciones con el teclado, existen muchas formas de realizar simples tareas, tales como la carga de los registros.

Paso 4

Trate de cargar otros registros con datos. En particular, intente uno de los que no son pares de registros como IX, IY, SP o PC. Obsérvese que si usted entra dos dígitos hex y pulsa la tecla ST el registro de 16 bits es cargado con 00 a la izquierda

y el byte (dos dígitos) que usted ha entrado a la deredcha. Trate ahora de entrar cuatro dígitos hex, pongamos ABCD, y pulse STORE. ¿Qué sucede? ¿Puede usted explicar esto?

Observamos que ABCD aparece en el display de direcciones indicando que el registro de 16 bits ha sido cargado con dos nuevos bytes (a saber AB y CD). La explicación para estos "ceros a la izquierda" cuando solamente se almacenan dos dígitos, es la siguiente. SIEMPRE se almacenan cuatro dígitos (dos bytes) con la lámpara selectora en IX, IY, SP o HL. El Nanocomputador coloca dos ceros a la izquierda si usted solamente entra dos dígitos. ¿Qué sucede si usted entra un dígito y pulsa ST? ¿Tres dígitos y pulsa ST? Esto se llama *colocar ceros a la izquierda.* Evita el tener que pulsar los ceros que preceden al número.

EXPERIMENTO N.° 3

Propósito

El propósito de este experimento es el de demostrar la carga de datos en posiciones de memoria y utilizar la tecla INC para mostrar el contenido de sucesivas posiciones de memoria.

Paso 1

Puesto que estamos trabajando con posiciones de MEMoria y su contenido, posicionar la lámpara de selección en MEM. Como ya hemos mencionado antes, los cuatro dígitos del display de direcciones y del display de datos dan el contenido de esta posición. Vamos a mirar el contenido de la posición 0100. Para hacer esto entrar 0100 ó 100 mediante el teclado, pulsar a continuación LA, la tecla para cargar direcciones. ¿Qué es lo que usted observa?

Usted debe ver que aparece 0100 en el display de direcciones. En nuestro caso, el display de datos debe indicar 00. Esto es, en nuestro caso, la posición 0100 contiene 8 bits todos ellos colocados a 0, o sea el byte 00. Su dato puede ser distinto.

Paso 2

Intente almacenar AA en la posición 0100. Entre AA, pulse la tecla ST. ¿Qué ha sucedido?

El display de direcciones debe leer 0101 y, en nuestro caso, observamos que el display de datos indica 00.

¿Qué significa esto? Significa que AA se ha almacenado en la posición 0100 y que el computador está esperando que usted almacene un byte en la próxima posición (0101). El contenido actual de esta posición (0101) aparece en el display de datos (el nuestro lee 00). Intente almacenar (STore) BB, entrando BB y pulsando ST.

Paso 3

No confiemos demasiado en nuestro Nanocomputador. Debemos hacer una comprobación para asegurarnos de que la posición 0100 contiene AA y que 0101 contiene BB. ¿Cómo podemos hacer esto?

Para ver el contenido de 0100, necesitamos tener en el display 0100 como dirección. Esto necesita la tecla LA. Pulse 0100 ó 100 seguido por LA. Esperamos ver AA en el display de datos.

Paso 4

¿Cómo podemos ver la posición 0101? Usted puede entrar la dirección 0101 (ó 101) pulsando a continuación LA. Sin embargo, el Nanocomputador puede también INCrementar una posición de memoria de uno, evitando tener que volver a pulsar las direcciones. Pulse la tecla INC una vez ligeramente. ¿Qué es lo que usted ve?

El registro de direcciones es incrementado de uno y el registro de datos muestra BB. Pulse INC y mantenga la tecla pulsada. Como usted puede ver, las posiciones de memoria son mostradas secuencialmente.

Las teclas LA, ST e INC son las principales herramientas que usted utilizará en este libro para la carga de programas y su verificación. Usted verá en el próximo experimento lo importante que son estas teclas.

Paso 5

Vuelva a posicionar la lámpara selectora en cualquiera de las posiciones IR, AF, BC, DE, HL, IX, IY SP o PC, entre 11 y pulse LA. ¿Qué es lo que usted observa?

¡La lámpara roja de error se enciende! Le hemos dirigido a cometer una equivocación. Al utilizar la tecla LA con la lámpara selectora posicionada en cualquier parte que no sea MEM o I/O hemos intentado una operación ilegal. La única situación en la que se puede utilizar LA es para cargar la dirección del display con una dirección de MEMoria o un código de dispositivo de I/O.

Nota: Para anular la señal de error, simplemente empiece entrando la próxima instrucción por el teclado.

Paso 6

Mire si usted puede conseguir cargar el código de un dispositivo de un byte en el display de direcciones, apareciendo el contenido de la puerta de I/O automáticamente en el display de datos. El procedimiento es el siguiente: Posicione la lámpara de selección en I/O entre dos dígitos hex, y pulse LA.

EXPERIMENTO N.° 4

Propósito

El propósito de este experimento es cargar y ejecutar un programa muy simple en el microprocesador a toda velocidad y en modo paso a paso. Se demuestra el funcionamiento de las teclas SS, GO y RESET.

Paso 1

Al final de este paso, queremos que en las posiciones de memoria del Nanocomputador desde 0100 hasta 0105 aparezca lo siguiente:

Dirección	Contenido
0100	3E
0101	00
0102	3C
0103	C3
0104	02
0105	01

Para conseguirlo, posicione la lámpara selectora en MEM. Entre 0100 ó 100 y

pulse LA. La dirección del display debe indicar 0100 señalando que el Nanocomputador está preparado para empezar a cargar el programa en 0100.

Las siguientes pulsaciones cargarán el programa:

```
3E   pulse ST
00   pulse ST
3C   pulse ST
C3   pulse ST
02   pulse ST
01   pulse ST
```

Paso 2

Usted debe ahora verificar que ha cargado correctamente el programa. ¿Cómo puede hacer esto?

Utilice la tecla INC: primero muestre en el display la posición de memoria 0100 (utilice la tecla LA), pulse entonces la tecla INC para mostrar cada posición sucesiva de memoria. Asegúrese de que usted tiene el contenido de cada posición de memoria correcto.

Paso 3

Cargar el registro PC con la dirección inicial del programa, 0100. Para hacer esto, recuerde que usted primero debe posicionar la lámpara selectora en PC, entrar 0100 ó 100, y pulse ST.

Paso 4

Posicione la lámpara selectora en AF. Esto significa para el Nanocomputador que usted desea observar el contenido del par de registros AF mientras que el programa se ejecuta a baja velocidad. Pulse repetidamente la tecla SS. ¿Qué es lo que usted observa?

Con cada depresión de la tecla SS, usted debe ver que el contenido del registro A (el acumulador) se incrementa de uno. Mantenga la tecla SS apretada. El registro A continuará contando. Su Nanocomputador está ejecutando un programa ¡a baja velocidad!

Paso 5

Pulse ahora GO. ¿Qué ha sucedido?

Todas las lámparas y displays se han apagado. ¿Ha parado usted el Nanocomputador? ¡NO! Ahora está ejecutando el programa a toda velocidad. Usted puede detenerlo pulsando la tecla RESET. ¿Qué significa toda velocidad? El acumulador se estaba incrementando más de cien mil veces cada segundo. Nótese que inmediatamente después de pulsar la tecla RESET, el display del Nanocomputador se enciende con la lámpara de selección en la posición PC. Siempre que el Nanocomputador "queda obscuro" como usted acaba de observar, usted puede recuperar el control pulsando RESET o BREAK. Usted acaba de ejecutar su primer programa en el microcomputador.

EXPERIMENTO N.º 5

Propósito

El propósito de este experimento es el de investigar dos rutinas utilitarias que proporciona SGS-ATES con el sistema operativo del Nanocomputador en memoria de sólo lectura. Las dos rutinas utilitarias hacen un test de la memoria de lectura/escritura y el software/hardware del teclado/display.

Paso 1

Puesto que los programas que usted ejecutará en este experimento residen en memoria de sólo lectura, no hay necesidad de cargar manualmente ningún byte, como usted hizo anteriormente.

Vamos a examinar primeramente el test de la memoria de lectura/escritura. Este test comprende dos partes. La primera comprueba las posiciones de memoria de 0FB0 hasta 0FFF. Estas posiciones de memoria son utilizadas por el sistema operativo del Nanocomputador para guardar los datos tales como los bytes que se van a mostrar en el teclado. Cada posición de memoria se comprueba escribiendo ceros dentro de ella y leyéndola para ver si todavía están allí los ceros. Este no es un test exhaustivo, pero asegura alguna funcionalidad en cada posición de memoria. El test de las posiciones 0FB0 hasta 0FFF ocurre automáticamente cuando se pulsa la tecla RESET. Si no se detecta error en el test, el Nanocomputador muestra que está preparado encendiendo la lámpara PC y mostrando el contenido del registro PC a

la izquierda y el contenido de la posición de memoria señalada por el registro PC a la derecha.

Pulse la tecla RESET varias veces. Afortunadamente ¡usted no detecta errores!

Paso 2 (opcional)

Este paso NO es recomendado para los estudiantes que no tiene RAM dinámicas 4027 extras, porque estos chips son muy sensibles a la electricidad estática y se estropean fácilmente si se quitan de su zócalo o se reemplazan incorrectamente.

Antes de realizar el paso 2, le recomendamos que lea el apéndice E que trata de las precauciones para manipular dispositivos MOS.

Vamos a engañar al Nanocomputador haciéndole creer que tiene algunas posiciones de memoria estropeadas. En primer lugar QUITE LA ALIMENTACION AL NANOCOMPUTADOR. ¡Esto es crítico! Usted puede estropar el computador, si deja la tensión conectada. Con la tensión de alimentación desconectada, y refiriéndose al diagrama de la figura 4-7, remueva cuidadosamente el chip de memoria

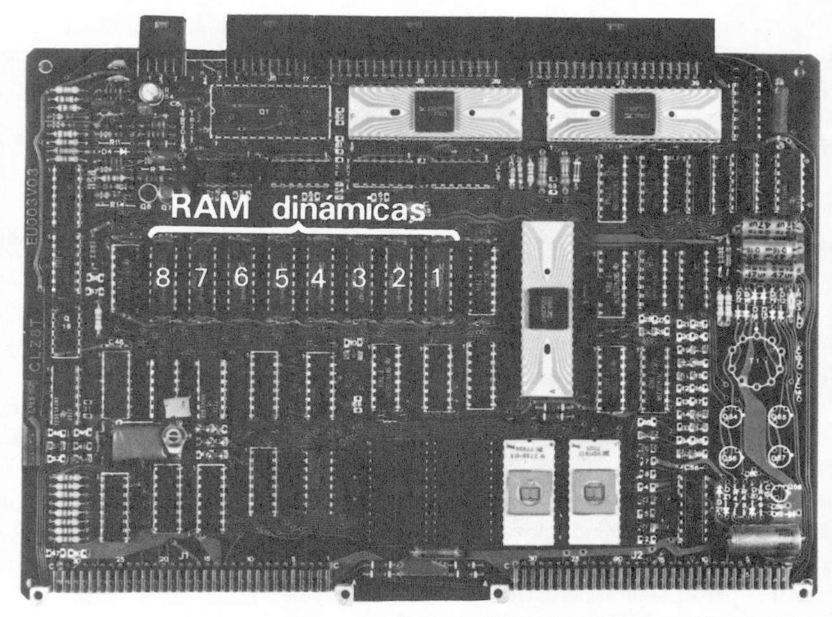


Fig. 4-7.– Números de chips RAM y números de display de dígitos KBD.

RAM N.° 1 de su zócalo de plástico. Esto se realiza más fácilmente con un pequeño destornillador para extraer el chip. *Tenga mucho cuidado de mantener las patillas del chip de memoria lo más rectas posible.* Con el chip de memoria fuera de su zócalo, vuelva a conectar el microcomputador y pulse la tecla RESET. ¿Qué es lo que usted observa?

Observamos que la lámpara ERR se enciende y en el display se lee

8 - - - - - - -

Esta salida poco usual es producida por la rutina de test de la memoria que ha detectado un error en el chip N.° 1. La presencia de un 8 (todos los segmentos encendidos) corresponde al chip que hemos sacado. Similarmente, quitando el chip de RAM N.° 3 obtendremos un 8 en la tercera posición (si el resto de los chips están en su posición correcta). Como que es muy sencillo el romper las patillas de los chips de los semiconductores: no le recomendamos que usted verifique experimentalmente la correspondencia entre la posición del chip y la posición del 8 en el display de error resultante.

QUITE LA TENSION AL NANOCOMPUTADOR y reemplace cuidadosamente el chip de memoria. Con el chip de RAM en su posición correcta vuelva a aplicar la tensión al Nanocomputador y pulse la tecla de RESET. Usted no debe encontrar ningún error.

Paso 3

La segunda parte del test de la memoria de lectura/escritura comprueba la RAM del usuario (figura 4-7), en las posiciones de memoria 0000 hasta 0FAF. Para cada posición de memoria se cargan alternativamente y se leen los siguientes bytes:

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
      . . .
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
```

Este procedimiento se llama "desplazar un bit por la memoria" y es una técnica comúnmente utilizada para hacer un test de la memoria. Este test de la memoria se puede ejecutar pulsando la siguiente secuencia de teclas:

1. Asegurarse de que la lámpara selectora está en la posición PC (utilice las teclas FLECHA DERECHA y FLECHA IZQUIERDA).

2. Entre la dirección asociada con la etiqueta MEMTUT (ver la Tabla Maestra de Símbolos en el apéndice F).
3. Pulse la tecla GO.

Usted acaba de ejecutar el programa que reside en la posición de memoria CONTST, el cual, es el programa de test de la memoria. El display debe quedar apagado y el computador muy atareado va desplazando los bits en la memoria. Esto continuaría eternamente. Solamente se iluminará el display si ocurre alguna de estas dos cosas:

1. Usted pulsa la tecla RESET o BREAK, lo cual termina el test y provoca el retorno al estado normal de funcionamiento con la lámpara de selección en la posición PC.
2. El test de la memoria detecta una posición de memoria defectuosa. En este caso el display mostrará lo siguiente:

   Dirección del byte erróneo en los cuatro dígitos que están más a la izquierda.
   El byte de datos escrito en los dos dígitos siguientes.
   El byte de datos leído en los próximos dos dígitos.

Naturalmente, el test falló porque los últimos dos pares de dígitos no fueron iguales.

Paso 4

El test del teclado/display comprueba todas las teclas menos dos (BREAK y RESET) y todos los LED (indicadores luminosos) de la unidad teclado/display del Nanocomputador. El programa empieza en la posición CONTST (ver la Tabla Maestra de Símbolos en el apéndice F). Así, para ejecutarlo:

1. Posicione la lámpara selectora en PC.
2. Entre la dirección asociada con la etiqueta CONTST.
3. Pulse la tecla GO.

¿Qué es lo que usted observa?

Observamos que las lámparas BRK y IR se han encendido. También se iluminó uno de los segmentos de cada uno de los displays de 7 segmentos. Para su conveniencia y facilidad de referencia, los segmentos están indicados con letras, como se muestra en la figura 4-8 y en la siguiente lista.

A continuación damos una correspondencia entre las filas de las teclas (numeradas desde abajo hacia arriba) y los LED y segmentos encendidos.

Fila de la tecla	LED iluminado	Segmento iluminado
1	BRK, IR	a
2	AF, I/O	b
3	BC, MEM	c
4	PC, DE	d
5	SP, HL	e
6	ERR, IX	f
7	IY, ARS	g

Las teclas BREAK y RESET no se pueden comprobar, puesto que al pulsar cualquiera de ellas termina el test. Verifique la correspondencia anterior pulsando varias teclas en cada fila horizontal de teclas en el teclado del Nanocomputador.

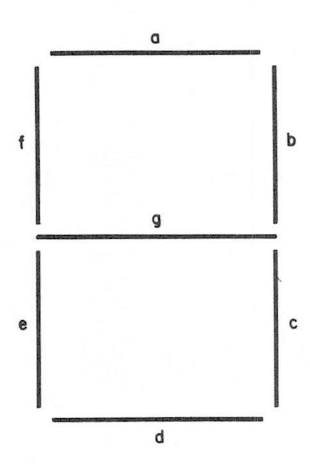


Fig. 4-8.– Esquema de numeración del display de siete segmentos.

5

Algunos programas simples del microcomputador Z-80

En este capítulo, usted cargará y ejecutará varios programas simples que utilizan las instrucciones del Z-80 discutidas en el capítulo 3.

OBJETIVOS

Al finalizar este capítulo, usted será capaz de hacer lo siguiente:

- Definir los términos *código binario, código hexadecimal, código ensamblador* y *lenguaje de alto nivel.*
- Explicar el funcionamiento de las siguientes instrucciones del Z-80: NOP; INC A; HALT; LD A, <B2>; LD (<B2> <B3>), A; y JP <B3> <B2>.
- Cargar y ejecutar programas simples del microprocesador Z-80 en el Nanocomputador.
- Ser capaz de leer y comprender los listados en lenguaje ensamblador del Z-80 que muestran la posición de memoria, código objeto, código fuente y comentarios.

REPASO DE VARIAS INSTRUCCIONES DEL Z-80

En el capítulo 3 discutimos las siguientes instrucciones del Z-80:

Código máquina hex	Código mnemónico	Operación
00	NOP	No operación
3C	INC A	Incrementar el acumulador de 1
76	HALT	Parar el microcomputador
3E <B2>	LD A,<B2>	Mover el byte de dato siguiente al acumulador
32 <B2> <B3>	LD K(<B3><B2>),A	Cargar el contenido del acumulador en la posición de memoria direccionada por los 2 siguientes bytes en esta instrucción de 3 bytes
C3 <B2> <B3>	JP <B3><B2>	Salto incondicional a la dirección de memoria dada en los dos bytes siguientes de esta instrucción de 3 bytes.

LENGUAJES DE PROGRAMACION Y LISTADOS

Usted leyó en el capítulo 2 que un programa consiste en una serie de instrucciones y que las instrucciones se escriben de formas muy variadas: binario, hexadecimal, código mnemónico y palabras completas (lenguaje de alto nivel). Vamos a examinar estas formas más detalladamente.

Código binario

Este es el verdadero lenguaje del Z-80. Eventualmente, todas las instrucciones del programa deben ser expresadas de esta forma para que el Z-80 las pueda entender. Si las personas utilizaran normalmente este lenguaje para expresarse, no sería necesario presentar de otra forma las instrucciones para el computador. Cada forma subsiguiente, hexadecimal, mnemónico y lenguaje de alto nivel, se acerca más y más a las personas y menos al computador. Obviamente, se paga un precio para cada nivel sucesivo de conveniencia para las personas, principalmente el tiempo que se necesita para realizar la traducción a bits, y espacio de memoria que se necesita para albergar el programa que realice la traducción.

Vamos a mirar a un programa simple, escrito en código binario. El siguiente programa suma los dos números en las posiciones de memoria 0160H y 0161H y guarda la suma en la posición 0162H:

Listado de un programa en código binario

```
0 0 1 1 1 0 1 0
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 1 1 1
0 0 1 1 1 0 1 0
```

```
0 1 1 0 0 0 0 1
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0
0 1 1 0 0 0 1 0
0 0 0 0 0 0 0 1
```

Esto puede parecer agradable a su Nanocomputador, pero resulta poco natural para usted.

Código hexadecimal

Esta forma de representación de las instrucciones es una mejora con respecto al método binario puesto que abrevia cada grupo binario de 8 bits a dos dígitos hexadecimales. Vamos a listar el programa precedente utilizando el código hexadecimal:

Listado de un programa en código hexadecimal

```
3A
60
01
47
3A
61
01
80
32
62
01
```

Esto es ciertamente una mejora desde el punto de vista de las personas, pero un Z-80 no sería capaz de interpretar este programa tal como está.

Se necesita un *cargador hexadecimal* para que tome el anterior listado hexadecimal y lo convierta en el código binario que el Z-80 puede entender. El sistema operativo del Nanocomputador contiene un cargador hexadecimal que detecta cuando se ha pulsado una tecla hexadecimal (0-F) y convierte el código hex a código binario para guardarlo en memoria de lectura/escritura.

¿Cuáles son las ventajas y desventajas de los cargadores hex? Las ventajas son:

1. Mayor facilidad en la comunicación programador/máquina, porque solamente se necesitan dos dígitos en lugar de los ocho que se utilizarían en código binario.
2. Aumento de la eficiencia del programador, debido a que los errores se detectan y corrigen más fácilmente.

Sin embargo, las desventajas son:

1. El cargador hex –que también es un programa– debe residir en la memoria para que se pueda interpretar el código hex (es decir, ser convertido a una representación binaria) y almacenado.
2. El proceso de conversión toma tiempo.

Hoy en día, los programadores son más caros que los computadores, y por esto los factores humanos son considerados en muchos casos como más importantes, y seguramente aumentarán su importancia relativa.

Código mnemónico

Esta forma de representar las instrucciones nos lleva un paso más lejos del computador y más cerca del programador. El código mnemónico utiliza caracteres alfabéticos para describir las instrucciones. Por ejemplo LD B,A es el código mnemónico para la instrucción codificada 47 en hex. Esta instrucción hace que el contenido del registro A se cargue en el registro B. Claramente, la representación mnemónica es más sugestiva de la operación de que se trata (al menos desde un punto de vista humano). A continuación se da el listado en código mnemónico para el programa de suma, descrito anteriormente.

Listado de un programa mnemónico o listado en ensamblador

```
LD      A, (0160H)
LD      B, A
LD      A, (0161H)
ADD     A, B
LD      (0162H),A
```

(El carácter "H" que sigue a la dirección indica que está expresado en HEX.) Obsérvese que:

a. Se utilizan los mnemónicos para describir las operaciones que va a realizar el Z-80. Por ejemplo, el mnemónico "LD" es una abreviación para "LOAD" que es una instrucción que mueve datos desde un origen a un destino y tiene la forma general:

   LD "destino", "origen"

   La primera instrucción Load carga el registro A o acumulador con el contenido de la posición de memoria 0160H.

b. Se han asignado nombres a los registros que están en un chip de la CPU del Z-80. Por ejemplo, los registros A y B (conocidos normalmente como 111 y 000 para el Z-80) se mencionan en el programa precedente.

Un programa escrito utilizando mnemónicos se llama un programa en *lenguaje ensamblador.* Los mnemónicos utilizados en el lenguaje ensamblador del programa anterior fueron desarrollados por Zilog Corporation cuando desarrollaron la CPU del Z-80, siendo también los mnemónicos recomendados por SGS-ATES. Sin embargo, SGS-ATES no puede obligar al usuario a utilizar estos mnemónicos y muchas otras compañías han desarrollado grupos equivalentes de mnemónicos que ellos piensan que son mejores. Para el Z-80, el conjunto de mnemónicos más utilizado y universal es el de SGS-ATES y Zilog, de forma que nosotros utilizaremos este conjunto en este libro.

Como usted bien puede imaginar, el proceso para trasladar un programa en lenguaje ensamblador a código binario es bastante complicado. Esto es cierto, pero el proceso es tan sistemático y repetitivo que puede ser programado e implementado en el mismo Z-80. El programa que acepta un listado en código ensamblador (llamado *código fuente*) y saca un programa codificado en binario (llamado *código objeto*) es llamado un ensamblador (*assembler*). SGS-ATES ha escrito un ensamblador para el Nanocomputador que está normalmente disponible en una cinta cassette ASS-Z o EPROM FR-Z. Para utilizar el ensamblador, usted necesita una versión mejorada del Nanocomputador, un teclado con caracteres ASCII (el alfabeto, números y caracteres especiales tales como el punto, la coma, punto y coma, etc.) para entrar los mnemónicos, y un dispositivo de display en ASCII como salida así como un mínimo de 16K bytes de memoria de lectura/escritura y dos cassettes de audio para grabar y escuchar. No supondremos que usted tiene una configuración de microcomputador tan sofisticada. Así, en este libro, realizaremos el ensamblado del programa a mano.

Por ensamblado a mano, entendemos que el programador toma el conjunto de mnemónicos y los traduce uno por uno a código hex. El código hex se puede entrar al Nanocomputador cuyo cargador hex realiza la conversión final a código binario que puede entender el Z-80. El proceso de ensamblado a mano utiliza intensivamente referencias cruzadas de mnemónico a hex. Hay un conjunto excelente de referencias cruzadas en forma de matriz con las instrucciones agrupadas por función similar. Todas ellas se presentarán en este libro.

A continuación se da un listado de un programa ensamblado a mano. Nótese que el código hex se llama aquí *código objeto,* y que el código ensamblador se llama *código fuente.* Obsérvese también que un punto y coma (;) separa cada comentario de su línea asociada de código fuente. El propósito de este punto y coma es el de notificar al ensamblador que debe de ignorar todo lo que sigue, es decir, que lo que

sigue solamente sirve para las personas. Además, los listados en ensamblador siguen estrictamente las convenciones mencionadas anteriormente acerca de las representaciones numéricas: los números decimales están seguidos por un punto (.), los números binarios aparecen sin notación especial.

Posición de memoria	Código objeto	Código fuente		Comentarios
0150	3A 60 01	LD	A,(0160H)	;A= contenido de la posición 0160
0153	47	LD	B,A	;Cargar B con A
0154	3A 61 01	LD	A,(0161H)	;A= contenido de la posición 0161
0157	80	ADD	A, B	;Sumar A y B
0158	32 62 01	LD	(0162H),A	;Guardar la suma en 0162

Así, se puede ver que el ensamblado a mano requiere la traducción de los mnemónicos y además la colocación del byte resultante en memoria de lectura/escritura. En el caso previo, el programa empieza en la posición 0150 y finaliza en 015A. Los comentarios son, desde luego, opcionales pero muy deseables. Esta es la forma en que todos los programas se escribirán en este libro.

Lenguajes de alto nivel

La última categoría de lenguajes de programación que discutiremos es llamada lenguajes de alto nivel. Los lenguajes de alto nivel están alejados un paso más del computador que los ensambladores. Típicamente, los lenguajes de alto nivel no necesitan que el programador conozca nada acerca de los registros o direcciones de memoria. Por el contrario, estos lenguajes están diseñados para permitir al programador que se concentre en el problema a resolver, en lugar de concentrarse en el computador. Por ejemplo, en un lenguaje de alto nivel tal como el FORTRAN, el programa para sumar dos números aparecería en la forma siguiente:

```
ANS = X + Y
```

Ejemplos de otros lenguajes de alto nivel son: COBOL, PL/1, ALGOL, SNOBOL, PASCAL, JOVIAL y muchos más. Cada uno de estos programas fuente necesita de grandes traductores llamados *compiladores* para convertir los programas a código objeto en binario.

En un Nanocomputador ampliado se puede implementar un lenguaje de alto nivel llamado *BASIC.* El BASIC de SGS-ATES es un lenguaje orientado a aplicaciones de control y está disponible en 8K de PROM/ROM (BAS-Z). En el momento

en que se entra cada sentencia en lenguaje BASIC, el traductor BASIC interpreta la sentencia, la convierte a (muchos) bytes de código objeto en binario y lo ejecuta inmediatamente. Esto no es parecido a un compilador FORTRAN o COBOL o el ensamblador del Z-80 los cuales esperan hasta que todo el programa se ha entrado antes de empezar la traducción. Por esta razón al traductor BASIC se le llama un *INTERPRETE*. La versión en BASIC de nuestro programa de suma es:

```
LET A1 = X + Y
```

Los lenguajes de alto nivel tienen dos ventajas importantes que a menudo superan las desventajas de los grandes compiladores o intérpretes:

1. Los programas están "orientados al procedimiento" en mayor grado y orientados menos al computador.
2. Los programas escritos en un lenguaje de alto nivel para un computador pueden funcionar a menudo en otro computador con pocos o ningún cambio, que es una propiedad llamada *portabilidad*. Esto casi nunca es posible bien sea con lenguaje ensamblador o utilizando los códigos hex, debido simplemente a que están demasiado relacionados a la máquina.

PROGRAMACION EN LENGUAJE ENSAMBLADOR

Como hemos mencionado anteriormente, en esta serie de capítulos programaremos en lenguaje ensamblador y ensamblaremos a mano los programas para obtener el código hex que será cargado en el Nanocomputador. Además de traducir los códigos mnemónicos a código hex, el ensamblado a mano requiere posicionar el programa del microcomputador en alguna parte de la memoria disponible de lectura/escritura de su Nanocomputador. Por favor, obsérvese la primera columna del listado en ensamblador dado en la última sección. Esta columna, con la cabecera "Posición de memoria", indica la dirección de memoria del primer byte en aquella línea de programa. Si esta línea está ocupada por más de un byte, las próximas direcciones en secuencia contienen estos bytes, empezando la próxima línea de programa en la próxima posición de memoria.

Vamos ahora a tratar de ejecutar algunos programas simples en el Nanocomputador.

INTRODUCCION A LOS EXPERIMENTOS

Los siguientes experimentos le permitirán ejecutar varios programas simples que

están descritos en detalle para cada experimento. Esto le dará experiencia en la carga y ejecución de los programas en el microprocesador, así como enseñarle algunos rudimentos de la programación.

Los experimentos que usted realizará se pueden resumir de la siguiente forma:

Experimento N.°	Comentarios
1	Demuestra la ejecución de las instrucciones NOP, HALT y LD A, <B2>.
2	Demuestra la ejecución de las instrucciones INC A y JP <B3> <B2> en un simple bucle de programa.
3	Demuestra la ejecución de un programa con un solo bucle, y las instrucciones INC B e INC C.
4	Demuestra las instrucciones LD (<B2>,<B3>), A; son instrucciones que se utilizan para colocar el contenido de la memoria a un determinado valor.
5	Demuestra la ejecución de un programa de suma que se utilizó como un ejemplo en la sección de los lenguajes de programación y listados.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar la ejecución de tres instrucciones del Z-80: NOP, HALT y LD A, <B2>.

Programa N.° 1

Posición de memoria	Código objeto	Código fuente	Comentarios
0100	3E BB	LD A, BBH	;Cargar el acumulador con BB
0102	76	HALT	;Parar el microcomputador

Programa N.° 2

0103	3E BB	LD A, BBH	;Cargar el acumulador con BB
0105	00	NOP	;No operación
0106	3E FF	LD A, FFH	;Cargar el acumulador con FF
0108	76	HALT	;Paro

Paso 1

Dar tensión al microcomputador y pulsar la tecla de RESET varias veces para inicializar la CPU del Z-80. Posicionar la lamparita del selector en MEM, entrar 100, y pulsar LA. Usted está preparado para empezar a cargar el programa en la memoria empezando en la posición 0100. Entrar el programa N.° 1 entrando sucesivamente los códigos objeto y pulsando cada vez la tecla ST.

Paso 2

Volver a comprobar que el programa N.° 1 ha sido bien cargado en la memoria, posicionando el indicador de la memoria en 0100 (entrar 0100 seguido por LA) y pulsar la tecla INC.

Paso 3

Antes de ejecutar este programa, vamos a reflexionar en lo que esperamos que ocurra. Por favor escriba en el siguiente espacio cuál debe ser el contenido del acumulador después de que haya terminado la ejecución del programa.

Una vez dada su predicción, vamos a proceder a averiguar si está en lo cierto. Primero vamos a mirar el acumulador y ver su contenido: posicione la lamparita selectora en AF. ¿Qué es lo que indican los dos dígitos más a la izquierda del display?

Paso 4

Ejecutar el programa empezando en 0100: posicionar la lámpara selectora en la posición PC, entrar 0100, ST, pulsar a continuación GO. ¿Qué sucede?

El Nanocomputador inmediatamente ¡se queda apagado! El computador está en el estado HALT.

Paso 5

Pulsando la tecla BREAK, el Nanocomputador volverá a entrar en actividad. El

computador inmediatamente "se espera" con la lámpara indicadora en la posición PC mostrando el contenido del contador de programa o registro PC y que es 0103. Esta es la posición de memoria de la próxima instrucción que se debe ejecutar. Sin embargo, debido a que la instrucción en la posición 0102 es HALT (76), no se alcanza la próxima instrucción.

Paso 6

Comprobar el contenido del acumulador para ver si su predicción en el paso 3 fue correcta: Mueva la lámpara del selector a AF. ¿Qué es lo que vé como contenido del acumulador?

Esperamos que usted vea BB. La primera instrucción del programa N.° 1 le dice al microcomputador que ponga BB en el registro A, el acumulador. La próxima instrucción que es la que para al microcomputador, no afecta al registro A, o sea que después de la ejecución del programa, el contenido de A debe ser BB.

Paso 7

Cambiar el contenido de A a 00 (00, 2ND, ST) y ejecute a continuación al programa N.° 1 en modo paso a paso, observando el acumulador: posicione la lámpara selectora en PC, entre 0100, pulse ST, posicione la lámpara selectora en AF, y pulsar SS de nuevo. ¿Qué ocurre?

Inmediatamente el contenido del registro A cambia de 00 a BB.

Paso 8

Pulse de nuevo SS. Usted observará que no sucede nada. El dispositivo de paso a paso no permite que el computador se pare. Así en modo paso a paso, el mando HALT no tiene efecto.

Paso 9

Cargar y comprobar el programa N.° 2 empezando en la posición 0103. ¿Cuál debe ser el contenido del registro A al finalizar la ejecución de este programa?

Paso 10

Ejecutar el programa N.° 2. Asegúrese de nuevo de reanimar al Nanocomputador utilizando la tecla BREAK (no utilice RESET puesto que cambia el contenido de todos los registros). Comprobar el registro A.

Esperamos que lea FF como su contenido. Así, lo que ha sucedido es que el valor original BB ha sido cambiado con FF.

Paso 11

Vamos a ejecutar este programa en modo paso a paso para observar el efecto de la instrucción NOP en la posición 0105. Colocar el PC en 0103 y entonces posicionar la lámpara de selección en AF para observar la ejecución en paso a paso. Pulsar SS: El registro A pasa inmediatamente a BB. Pulsar SS: no sucede nada. Esta es la ejecución de la instrucción NOP. Si usted posiciona la lámpara de selección en PC verá que la dirección del display es 0106 indicando que 0105, la instrucción NOP, se ha terminado de ejecutar y que a continuación viene la instrucción 0106. Posicione el selector hacia atrás en AF y pulse de nuevo SS. En el acumulador se lee inmediatamente FF mostrando el efecto de la instrucción LD A, FFH.

Así, usted puede ver de los pasos anteriores que la instrucción NOP tiene solamente un efecto en un programa y es el de que ésta hace que el programa no haga nada durante este paso.

EXPERIMENTO N.° 2

Propósito

El proposito de este experimento es el de demostrar la ejecución de las instrucciones INC A y JP <B3><B2> en un simple bucle de programa.

Programa N.° 3

Posición de memoria	Código objeto	Código fuente	Comentarios
0109	3E FF	LD A, FFH	;Cargar el acumulador con FF
010B	3C	INC A	;Incrementar el acumulador
010C	C3 0B 01	JP 010BH	;Saltar a la dirección 010B

Paso 1

Obsérvese que la instrucción INC A es una instrucción de un byte que le dice al computador que le sume 1 al contenido del acumulador. La instrucción JP hace que el control se transfiera de forma incondicional a la dirección dada por los dos bytes que siguen a continuación del código de operación C3. Obsérvese que los dos bytes de dirección aparecen con el byte de dirección baja LO en primer lugar y que el byte de dirección aparece en segundo lugar.

Paso 2

Cargar y verificar el programa N.° 3.

Paso 3

Vamos a examinar el programa N.° 3 para anticipar que es lo que va a hacer. El programa N.° 3 inicializa el acumulador a FF e incrementa A de 1. La tercera instrucción provoca un salto incondicional hacia atrás a la posición 010B, es decir la instrucción INC A. De esta forma se incrementa el acumulador seguido por un salto hacia atrás a la instrucción INC A. El efecto de todo esto es que la instrucción INC A se ejecuta repetidamente hasta que alguien pare el "bucle", desconectando el computador, o pulsando RESET o BREAK.

Paso 4

Vamos ahora a ejecutar el programa en modo paso a paso para ver que le sucede al acumulador (A) y al contador de programa (PC). En primer lugar vamos a observar el acumulador, de forma que, después de cargar el PC con la dirección de inicio del programa, 0109, mover la lámpara de selección a la posición AF. Obsérvese que los dos dígitos más a la izquierda del display representan el contenido del registro A. Pulsar SS una vez más. ¿Qué le sucede al registro A?

Observamos que su contenido se convierte inmediatamente en FF.

Paso 5

La próxima instrucción indica incrementar A. ¿Cuál debe ser el contenido del acumulador después del próximo paso?

Pulse de nuevo SS para verificar su conjetura. Usted debe observar que el contenido del A ha pasado a 00. ¿Ha observado también algún cambio en el registro F? Esto lo discutiremos más adelante.

Paso 6

Mantenga la tecla SS pulsada durante un tiempo. Usted debe observar que el contenido del registro se incrementa a medida que se va ejecutando la instrucción INC A repetidamente.

Paso 7

Posicione ahora la lámpara de selección en PC. Mantenga la tecla SS pulsada y observe lo que sucede al contenido del registro PC. Escriba a continuación lo que usted observe:

Hemos observado que el registro PC va alternando entre 010B y 010C.

EXPERIMENTO N.° 3

Propósito

Demostrar la ejecución de un programa con un simple bucle y las instrucciones INC B e INC C.

Programa N.° 4

Posición de memoria	Código objeto	Código fuente	Comentarios
010F	04	INC B	;Incrementar el registro B
0110	C3 20 01	JP 0120H	;Saltar a la posición 0120
0120	C3 30 01	JP 0130H	;Saltar a la posición 0130
0130	04	INC B	;Incrementar el registro B
0131	0C	INC C	;Incrementar el registro C
0132	C3 0F 01	JP 010FH	;Saltar a la posición 010F

Paso 1

Vamos a hacer algunas observaciones acerca del listado del programa N.° 4.

a. En primer lugar obsérvese que la instrucción tal como JP 0130H se traduce a una serie de tres bytes hex:

C3	Código de operación de la instrucción de salto
30	Byte de dirección baja (LO)
01	Byte de dirección alta (HI)

Existe siempre la tentación de traducir la instrucción con los bytes de dirección en el orden inverso. Todos los programadores caen en este error por lo menos una vez. Todo lo que podemos hacer es mantenerle prevenido.

b. En el programa aparecen dos nuevas instrucciones, INC B e INC C. Estas hacen que el computador le sume 1 a los registros B y C, respectivamente.

Paso 2

Cargar y verificar el programa N.° 4. Obsérvese que usted debe cargar solamente los bytes especificados puesto que no se utilizan nunca otras posiciones de memoria entre 010F y 0134. La carga de este programa requiere que usted tenga una excelente comprensión de las teclas LA y ST y de las sutilidades de su empleo.

Paso 3

Cuando usted esté convencido de que ha cargado correctamente el programa N.° 4, estudie cuidadosamente los códigos mnemónicos del programa para determinar exactamente lo que está haciendo el programa.

Vamos a describir con palabras y después mediante un dibujo, qué es lo que hace el programa.

1. El primer paso del programa incrementa el registro B, es decir, añade 1 a su contenido actual.
2. El próximo paso es un salto incondicional a la posición 0120. Esto significa que este paso le indica al computador que vaya a la posición 0120 para su próxima instrucción.
3. Ahora el computador alcanza 0120 y lee allí la instrucción. ¡Otro salto incondicional! ¿A dónde vamos a ir esta vez? 0130.
4. Así, aquí estamos en 0130, ¿y ahora qué? El programa dice incrementar B, y se le añade 1 al registro B.
5. La próxima instrucción dice incrementar el registro C. Así se le suma 1 a C.
6. La próxima instrucción es un salto, de forma que vamos a la posición 010F.

Obsérvese que 010F es la instrucción que se había ejecutado en el paso 1. Así hemos recorrido un ciclo: Los pasos 1-6 se ejecutarán repetidamente hasta que intervenga alguna fuerza exterior.

Vamos a utilizar el diagrama de la figura 5-1.

El resultado final de estos incrementos y saltos es que el registro B es incrementado dos veces más a menudo que el registro C.

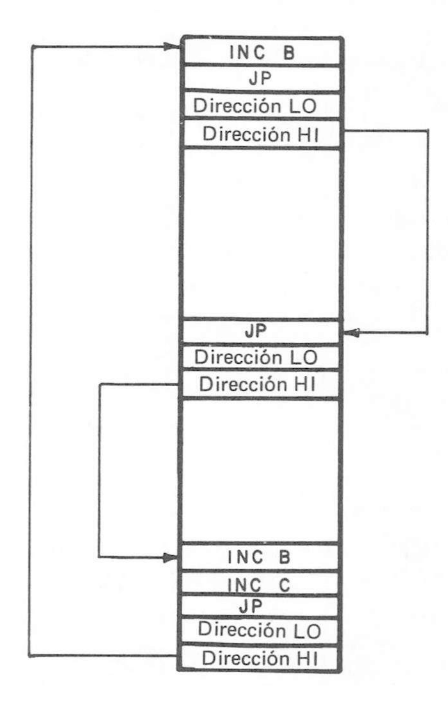


Figura 5-1.

Paso 4

Vamos ahora a ejecutar el programa N.º 4 en modo paso a paso. Deseamos observar que le sucede al registro PC y al par de registros BC durante la ejecución. Primeramente, cargar el par de registros BC con ceros para inicializar ambos registros (posicione la lámpara selectora en BC, entrar 00, 2ND, ST para inicializar B, y entrar 00, ST, para inicializar C). Ahora cargar el registro PC con 010F, y dejando

la lámpara de selección en PC, pulsar SS. ¿Cuántas veces deberá usted pulsar SS antes de que PC lea 0120, 0130, 010F por la segunda vez?

Su respuesta debe ser 2, 3 y 6, respectivamente. Así nosotros tenemos un bucle de programa de seis pasos. ¿Cuándo terminará este bucle? Nunca, hasta que lo paremos pulsando RESET o BREAK.

Paso 5

Mantenga apretada la tecla SS y observe la gama recorrida por las seis direcciones del PC. Escribir estas direcciones en el siguiente espacio.

Observamos las siguientes direcciones del PC:

010F
0110
0120
0130
0131
0132

Paso 6

Vamos ahora a observar el par de registros BC durante la ejecución paso a paso del programa N.° 4. Dijimos en el paso 3, que el registro B se incrementaría dos veces por cada vez que lo hiciera el registro C. Vamos a ver si éste es el caso. Lea el par de registros BC y escriba lo que vea en el siguiente espacio. (Asegúrese que usted está al principio de nuevo ciclo, es decir, PC = 010F.)

Observamos 04 02. (Dependiendo del tiempo en que se ha mantenido apretada la tecla SS en el paso 5, sus resultados pueden ser diferentes. Sin embargo, el primer dígito debe ser el doble del segundo.) Esto es, si B ha sido incrementado desde 00 a 04 mientras que C ha sido incrementado de 00 a 02. Esto debe confirmar nuestra suposición. Para asegurarnos, empiece pulsando SS y escriba a continuación sus observaciones:

Observamos lo siguiente:

Durante el paso a paso 1: B se incrementó
Durante el paso a paso 4: B se incrementó de nuevo
Durante el paso a paso 5: C se incrementó.

Los pasos 2, 3 y 6 están ocupados con las instrucciones de salto.

Paso 7

Mantenga la tecla SS apretada durante un cierto tiempo. Observamos la siguiente secuencia de datos en el display:

08 04	0C 05	0E 07
09 04	0C 06	0F 07
0A 04	0D 06	10 07
0A 05	0E 06	10 08
0B 05		

Paso 8

Vamos a hacer una última observación acerca de este programa. Las primeras dos instrucciones de salto pueden ser reemplazadas por una sola instrucción de salto. ¿Cuál es?

La respuesta es JP 0130H. La razón es que el primer JP siempre provoca un salto a la instrucción situada en 0120, y éste un salto a 0130.

EXPERIMENTO N.° 4

Propósito

El propósito de este experimento es demostrar la instrucción LD (<B3> <B2>), A que sirve para colocar el contenido de la memoria a un determinado valor.

Programa N.° 5

Posición de memoria	Código objeto	Código fuente	Comentarios
0136	3E 11	LD A,11H	;Cargar A con 11
0138	32 45 01	LD (0145H),A	;Cargar la posición de memoria 0145 con el contenido de A
013B	76	HALT	;Paro

Paso 1

La nueva instrucción es LD (0145H), A que está en la dirección 0138. Esta instrucción copia el contenido del registro A en la posición de memoria 0145. Obsérvese que al igual que en la instrucción de salto, la dirección está formada por el segundo y tercer byte de esta instrucción LD con el byte LO precediendo al byte HI.

Paso 2

Cargue y verifique el programa N.° 5.

Paso 3

Compruebe el contenido actual de la posición 0145 y escríbalo en el siguiente espacio.

Observamos 00.

Paso 4

Ejecute el programa anterior a toda velocidad o a la velocidad de paso a paso (SS). Si usted ejecuta el programa a toda velocidad, devuelva el control al sistema operativo del Nanocomputador pulsando BREAK para asegurarse que los registros y la memoria quedan protegidos. Mire el contenido del acumulador y de la posición de memoria 0145. ¿Qué contienen?

Ambos deben contener 11.

Paso 5

Escriba, cargue y ejecute un programa, empezando en la posición 013C que guarde 22 en la posición de memoria 0146. Usted puede comprobar si el programa funciona mirando a la posición 0146 después de haber ejecutado el programa. Hemos escrito una respuesta aceptable en el siguiente espacio. Su respuesta puede, desde luego, diferir algo de la nuestra.

Respuesta

Posición de memoria	Código objeto	Código fuente	Comentarios
013C	3E 22	LD A,22H	;Cargar A con 22H
013E	32 46 01	LD (0146H),A	;Cargar la posición de memoria 0146 con el contenido de A
0141	76	HALT	;Paro

EXPERIMENTO N.º 5

Propósito

El propósito de este experimento es el de demostrar la ejecución de un programa de suma que se utilizó como un ejemplo en la sección de este capítulo sobre los lenguajes de programación y listados.

Programa N.º 6

Posición de memoria	Código objeto	Código fuente	Comentarios
0150	3A 60 01	LD A,(0160H)	;A = contenido de la posición 0160
0153	47	LD B,A	;Cargar el registro B con A
0154	3A 61 01	LD A,(0161H)	;A = contenido de la posición 0161
0157	80	ADD A, B	;Sumar B a A, guardar el resultado en A
0158	32 62 01	LD (0162H),A	;Guardar la suma en la posición 0162
015B	76	HALT	

Paso 1

Obsérvese que en este programa se han introducido varias instrucciones nuevas. Estas instrucciones se escriben y explican a continuación. Todas ellas se explicarán en detalle en los siguientes apartados.

Código objeto	Código mnemónico	Operación
3A <B2> <B3>	LD A, (<B3><B2>)	Cargar el contenido de la posición de memoria <B3><B2> dentro del acumulador.
47	LD B,A	Cargar el contenido del acumulador dentro del registro B.
80	ADD A, B	Suma el contenido del registro B al contenido del registro A guardando el resultado de la suma en el registro A.

Paso 2

Cargar y verificar el programa N.º 6.

Paso 3

Vamos ahora a comprobar el programa, sumando 2 y 3. Para hacer esto debemos

guardar 2 en la posición 0160 y 3 en la posición 0161 (podríamos guardar 2 en la posición 0161 y 3 en la posición 0160, esto no importa). De acuerdo con esto se deben guardar los valores hex correspondientes a 02 y 03. Cargue ahora el registro PC con 0150 y pulsar GO.

Paso 4

Pulsar BREAK para devolver el control al sistema operativo del Nanocomputador y entonces mire el contenido del registro A y de la posición de memoria 0162. ¿Qué es lo que ve usted?

Esperamos ver 05 y esto es desde luego lo que encontramos.

Paso 5

Ejecutar el programa en modo paso a paso observando el contenido de los registros A y B.

Paso 6

Intente sumar otro par de números. Para no tener problemas asegúrese que los dos números dados no sumen un mayor número que FFH o 255 (base 10), la capacidad de un byte de 8 bits.

Paso 7

Si usted se siente ambicioso, vea si puede deducir lo que sucede cuando la suma excede FFH (o 255, base 10).

REPASO

Las siguientes preguntas le ayudarán a repasar las instrucciones que usted ha aprendido en esta unidad.

1. Explicar que es lo que hacen cada una de las siguientes operaciones.

   a. 3E
      5B

b. C3
   A5
   03
c. 3C
d. 32
   E4
   1B
e. 76
f. 00

2. Proporcionar el código hex correcto para las siguientes operaciones refiriéndose al texto de este capítulo o al apéndice B.
   a. Salto a la dirección de memoria HI = 24 y LO = 53
   b. Guarde el contenido del acumulador en la posición de memoria HI = 02 y LO = 38
   c. Mover el byte de dato 92 inmediato al acumulador
   d. Incrementar el contenido del acumulador
   e. Parar el microcomputador
   f. No operación.

Respuestas

1. a. Mover el byte de dato 5B al acumulador
   b. Saltar a la dirección de memoria HI = 03 y LO = A5
   c. Incrementar el contenido del acumulador
   d. Guardar el contenido del acumulador en la posición de memoria HI = 1B y LO = E4
   e. Parar el microcomputador
   f. No operación.

2. a. C3
      53
      24
   b. 32
      38
      02
   c. 3E
      92
   d. 3C
   e. 76
   f. 00

6

Registros, memoria y transferencia de datos

En este capítulo, usted aprenderá algunas de las formas de transferir datos entre el chip del microprocesador Z-80 y la memoria, así como entre distintas posiciones de memoria. Usted dará su primera mirada a todo el conjunto de instrucciones del Z-80, el cual le puede impresionar al menos por su amplitud y complejidad. Se introducen las instrucciones JP NZ, INC y DEC de forma que usted pueda crear *bucles con tiempos de retardo* en sus programas.

OBJETIVOS

Al terminar este capítulo, usted será capaz de hacer lo siguiente:

- Entender qué significa la *decodificación de una instrucción*
- Citar el código binario de 3 bits asignado a cada uno de los registros de uso general
- Definir el término *modo de direccionamiento*
- Definir: *direccionamiento por registro*
direccionamiento inmediato
direccionamiento inmediato extendido
direccionamiento por registro indirecto
direccionamiento extendido
- Explicar las instrucciones de incrementar registro y decrementar registro

- Explicar las instrucciones LD para los anteriores modos de direccionamiento
- Explicar las instrucciones de transferencia de bloques
- Escribir un programa que tiene un bucle de retardo
- Escribir programas para varios tipos de transferencia de datos.

CONJUNTO DE INSTRUCCIONES DEL Z-80

El conjunto completo de instrucciones del Z-80 se da en las páginas siguientes en una forma similar a la sugerida por R. Baker para el microcomputador Intel 8080A. Esta forma de describir el conjunto de instrucciones apareció en primer lugar en *Byte,* una revista dedicada a los aficionados de los microcomputadores. En el capítulo 3, usted aprendió que el código de operación (op code) es el código para la operación específica que el microcomputador ejecuta. Con ocho bits de información es posible tener 2 elevado a 8 o 256 códigos de operación distintos; éstos se muestran en la tabla 6-1. Los cinco dígitos binarios de la izquierda son los primeros cinco dígitos binarios de un código de operación de 8 bits. Los restantes dígitos binarios se muestran en columnas en la parte alta de las mismas y se repiten en otras tres posiciones de la tabla. Los códigos de operación para las instrucciones de dos y tres bytes aparecen en las tablas 6-2 hasta la 6-5.

No le pediremos que memorice este conjunto de instrucciones. Nuestro propósito aquí es simplemente mostrarle el conjunto de instrucciones completo de forma que usted pueda consultarlo cuando aprenda una nueva instrucción. Por ejemplo, en el capítulo 3 usted aprendió las siguientes instrucciones:

Código objeto	Código fuente
00	NOP
32 <B2> <B3>	LD (<B3><B2>), A
3C	INC A
3E <B2>	LD A, <B2>
76	HALT
C3 <B2> <B3>	JP <B3><B2>

¿Puede usted encontrarlos en la tabla? (Ayuda: estas instrucciones tienen un código de operación de un byte.)

Las instrucciones del Z-80 presentadas en las tablas 6-1, 6-2, 6-3, 6-4 y 6-5, proporcionan una visión de sus capacidades. En particular, se puede ver rápidamente como el conjunto de instrucciones del 8080 de Intel forman la base para el lenguaje expandido del Z-80. De todos los códigos de un byte solamente 12 no se pueden utilizar en el 8080. Las nuevas instrucciones con un código de operación de un byte en el Z-80 son las instrucciones de intercambiar grupos de registros (EXX) y las instrucciones de "salto relativo" (JR) las cuales están encuadradas en la tabla 6-1. Los otros códigos de operación, que no se utilizan en el 8080A, son CB, DD, ED y

FD. Cada uno de éstos es siempre el primer byte de una instrucción con un código de operación de varios bytes utilizados solamente en el Z-80. Discutiremos cada uno de estos códigos de instrucción más adelante, pero primeramente explicaremos más acerca de las instrucciones con un código de operación de un byte.

Si usted examina los códigos de operación de un byte en la tabla 6-1, observará que los dos primeros bits determinan la forma general de funcionamiento. Todas las instrucciones de carga de un solo byte (LD) en la tabla 6-1 tienen 01 para sus dos primeros bits. Todas las instrucciones aritméticas y lógicas empiezan con los dos dígitos binarios 10. Con la excepción de las instrucciones de salto relativo (JR) (que empiezan todas ellas con 00), las instrucciones de bifurcación –saltos, llamadas y retornos– tienen sus primeros bits igual a 11. Cuando se determina la relación entre los bits individuales del código de operación y las operaciones que se realizan, usted está *decodificando* el código de operación. Esto es esencialmente lo que hace electrónicamente el decodificador de instrucción dentro del chip del microprocesador.

La decodificación anterior de los dos primeros bits del código de las instrucciones del Z-80 es solamente el principio. Si usted examina más de cerca las instrucciones de carga, que tienen los dos primeros bits igual a 01, observará lo siguiente:

1. Cada instrucción mueve el contenido de un registro a otro registro.
2. Cada registro tiene un código de tres bits que le está asociado.

Registro	Código binario	
B	000	
C	001	
D	010	
E	011	
H	100	
L	101	
(HL)	110	(ver nota)
A	111	(ver nota)

Nota: Recuerde que (HL) se refiere a la posición de memoria direccionada por el par de registros H y L. La letra A se refiere al registro acumulador, que hemos discutido previamente. Hablando estrictamente, (HL) no es un registro, pero a menudo nos referimos a ellos como si lo fueran, en ciertos contextos. Tendremos cuidado de especificar si deseamos incluir o no (HL) cuando utilizamos la palabra "registro".

Para ver como se aplica el código de tres bits en las instrucciones de carga, considere este byte de 8 bits:

0 1 1 1 1 0 1 0

Tabla 6-1. Códigos de operación de un byte del Z-80

	000	001	010	011	100	101	110	111
00 000	NOP	LD BC, <B3><B2>	LD (BC), A	INC BC	INC B	DEC B	LD B, <B2>	RLCA
00 001	EX AF, AF'	ADD HL, BC	LD A, (BC)	DEC BC	INC C	DEC C	LD C, <B2>	RRCA
00 010	DJNZ <B2>	LD DE, <B3><B2>	LD (DE), A	INC DE	INC D	DEC D	LD D, <B2>	RLA
00 011	JR <B2>	ADD HL, DE	LD A, (DE)	DEC DE	INC E	DEC E	LD E, <B2>	RRA
00 100	JR NZ, <B2>	LD HL, <B3><B2>	LD (<B3><B2>), HL	INC HL	INC H	DEC H	LD H, <B2>	DAA
00 101	JR Z, <B2>	ADD HL, HL	LD HL, (<B3><B2>)	DEC HL	INC L	DEC L	LD L, <B2>	CPL
00 110	JR NC, <B2>	LD SP, <B3><B2>	LD (<B3><B2>), A	INC SP	INC (HL)	DEC (HL)	LD (HL), <B2>	SCF
00 111	JR C, <B2>	ADD HL, SP	LD A, (<B3><B2>)	DEC SP	INC A	DEC A	LD A, <B2>	CCF

	000	001	010	011	100	101	110	111
01 000	LD B, B	LD B, C	LD B, D	LD B, E	LD B, H	LD B, L	LD B, (HL)	LD B, A
01 001	LD C, B	LD C, C	LD C, D	LD C, E	LD C, H	LD C, L	LD C, (HL)	LD C, A
01 010	LD D, B	LD D, C	LD D, D	LD D, E	LD D, H	LD D, L	LD D, (HL)	LD D, A
01 011	LD E, B	LD E, C	LD E, D	LD E, E	LD E,H	LD E, L	LD E, (HL)	LD E, A
01 100	LD H, B	LD H,C	LD H, D	LD H, E	LD H, H	LD H, L	LD H, (HL)	LD H, A
01 101	LD L, B	LD L, C	LD L, D	LD L, E	LD L, H	LD L, L	LD L, (HL)	LD L, A
01 110	LD (HL), B	LD (HL), C	LD (HL), D	LD (HL), E	LD (HL),H	LD (HL), L	HALT	LD (HL), A
01 111	LD A, B	LD A, C	LD A, D	LD A, E	LD A, H	LD A, L	LD A, (HL)	LD A, A

	000	001	010	011	100	101	110	111
10 000	ADD A, B	ADD A, C	ADD A, D	ADD A, E	ADD A, H	ADD A, L	ADD A, (HL)	ADD A, A
10 001	ADC A, B	ADC A, C	ADC A, D	ADC A, E	ADC A, H	ADC A, L	ADC A, (HL)	ADC A, A
10 010	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
10 011	SBC A, B	SBC A, C	SBC A, D	SBC A, E	SBC A, H	SBC A, L	SBC A, (HL)	SBC A, A
10 100	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
10 101	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
10 110	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
10 111	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A

	000	001	010	011	100	101	110	111
11 000	RET NZ	POP BC	JP NZ, <B3><B2>	JP <B3><B2>	CALL NZ, <B3><B2>	PUSH BC	ADD A, <B2>	RST 0
11 001	RET Z	RET	JP Z, <B3><B2>	ver nota 1	CALL Z, <B3><B2>	CALL <B3><B2>	ADC A, <B2>	RST 8
11 010	RET NC	POP DE	JP NC, <B3><B2>	OUT (<B2>), A	CALL NC, <B3><B2>	PUSH DE	SUB <B2>	RST 10H
11 011	RET C	EXX	JP C, <B3><B2>	IN A, (<B2>)	CALL C, <B3><B2>	ver nota 1	SBC A, <B2>	RST 18H
11 100	RET PO	POP HL	JP PO, <B3><B2>	EX (SP), HL	CALL PO, <B3><B2>	PUSH HL	AND <B2>	RST 20H
11 101	RET PE	JP (HL)	JP PE, <B3><B2>	EX DE, HL	CALL PE, <B3><B2>	ver nota 1	XOR <B2>	RST 28H
11 110	RET P	POP AF	JP P, <B3><B2>	DI	CALL P, <B3><B2>	PUSH AF	OR <B2>	RST 30H
11 111	RET M	LD SP, HL	JP M, <B3><B2>	EI	CALL M, <B3><B2>	ver nota 1	CP <B2>	RST 38H

NOTA: Los bytes CB, DD, ED y FD no aparecen en el conjunto de instrucciones del 8080A. En el conjunto de instrucciones del Z-80, siempre aparecen como el primer byte del código de operación de una instrucción multibyte.

Tabla 6-2. Códigos de operación de dos bytes del Z-80: Byte 1 = CB

	000	001	010	011	100	101	110	111
00 000	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A
00 001	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
00 010	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A
00 011	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
00 100	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A
00 101	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
00 110								
00 111	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
01 000	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A
01 001	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
01 010	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A
01 011	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
01 100	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A
01 101	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
01 110	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A
01 111	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A

	000	001	010	011	100	101	110	111
10 000	RES, 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A
10 001	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
10 010	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A
10 011	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
10 100	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A
10 101	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
10 110	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A
10 111	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
	000	001	010	011	100	101	110	111
11 000	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A
11 001	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
11 010	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A
11 0.1	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
11 100	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A
11 101	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
11 110	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A
11 111	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

NOTA: CB es el primer byte de todos los códigos de operación de dos bytes para la instrucción anterior. La posición de una instrucción en la tabla determina el segundo byte del código de operación.

Tabla 6-3. Códigos de operación de dos bytes del Z-80: Primer byte = DD

	000	001	010	011	100	101	110	111
00 000								
00 001		ADD IX,BC						
00 010								
00 011		ADD IX,DE						
00 100		LD IX,<B4><B3>	LD(<B4><B3>),IX	INC IX				
00 101		ADD IX,IX	LD IX,(<B4><B3>)	DEC IX				
00 110					INC (IX+<B3>)	DEC (IX+<B3>)	LD (IX+<B3>),<B4>	
00 111		ADD IX,SP						
	000	001	010	011	100	101	110	111
01 000							LD B,(IX+<B3>)	
01 001							LD C,(IX+<B3>)	
01 010							LD D,(IX+<B3>)	
01 011							LD E,(IX+<B3>)	
01 100							LD H,(IX+<B3>)	
01 101							LD L,(IX+<B3>)	
01 110	LD (IX+<B3>),B	LD (IX+<B3>),C	LD (IX+<B3>),D	LD (IX+<B3>),E	LD (IX+<B3>),H	LD (IX+<B3>),L		LD (IX+<B3>),A
01 111							LD A,(IX+<B3>)	

	000	001	010	011	100	101	110	111
10 000							ADD A,(IX+<B3>)	
10 001							ADC A,(IX+<B3>)	
10 010							SUB (IX+<B3>)	
10 011							SBC A,(IX+<B3>)	
10 100							AND (IX+<B3>)	
10 101							XOR (IX+<B3>)	
10 110							OR (IX+<B3>)	
10 111							CP (IX+<B3>)	
	000	001	010	011	100	101	110	111
11 000								
11 001								
11 010								
11 011								
11 100		POP IX		EX (SP),IX		PUSH IX		
11 101		JP (IX)						
11 110								
11 111		LD SP,IX						

NOTA: Existen 70 instrucciones del Z-80 con códigos de operación de dos y tres bytes para el registro IY. Su definición es exactamente la misma que para la instrucción IX quedando reemplazado DD por FD como primer byte.

Tabla 6-4. Códigos de operación de tres bytes del Z-80: Byte 1 = DD, Byte 2 = CB

	000	001	010	011	100	101	110	111
00 000							RLC (IX+<B3>)	
00 001							RRC (IX+<B3>)	
00 010							RL (IX+<B3>)	
00 011							RR (IX+<B3>)	
00 100							SLA (IX+<B3>)	
00 101							SRA (IX+<B3>)	
00 110							SRL (IX+<B3>)	
00 111								
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
01 000							BIT 0,(IX+<B3>)	
01 001							BIT 1,(IX+<B3>)	
01 010							BIT 2,(IX+<B3>)	
01 011							BIT 3,(IX+<B3>)	
01 100							BIT 4,(IX+<B3>)	
01 101							BIT 5,(IX+<B3>)	
01 110							BIT 6,(IX+<B3>)	
01 111							BIT 7,(IX+<B3>)	
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
10 000							RES 0,(IX+<B3>)	
10 001							RES 1,(IX+<B3>)	
10 010							RES 2,(IX+<B3>)	
10 011							RES 3,(IX+<B3>)	
10 100							RES 4,(IX+<B3>)	
10 101							RES 5,(IX+<B3>)	
10 110							RES 6,(IX+<B3>)	
10 111							RES 7,(IX+<B3>)	
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
11 000							SET 0,(IX+<B3>)	
11 001							SET 1,(IX+<B3>)	
11 010							SET 2,(IX+<B3>)	
11 011							SET 3,(IX+<B3>)	
11 100							SET 4,(IX+<B3>)	
11 101							SET 5,(IX+<B3>)	
11 110							SET 6,(IX+<B3>)	
11 111							SET 7,(IX+<B3>)	

Tabla 6-5. Códigos de operación de dos bytes del Z-80: Byte 1 = ED

	000	001	010	011	100	101	110	111
01 000	IN B,(C)	OUT (C),B	SBC HL,BC	LD (<B4><B3>),BC	NEG	RETN	IM0	LD I,A
01 001	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(<B4><B3>)		RETI		
01 010	IN D,(C)	OUT (C),D	SBC HL,DE	LD (<B4><B3>),DE			IM1	LD A,I
01 011	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(<B4><B3>)			IM2	
01 100	IN H,(C)	OUT (C),H	SBC HL,HL					RRD
01 101	IN L,(C)	OUT (C),L	ADC HL,HL					RLD
01 110			SBC HL,SP	LD (<B4><B3>),SP				
01 111	IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(<B4><B3>)				
	000	001	010	011				
10 000								
10 001								
10 010								
10 011								
10 100	LDI	CPI	INI	OUTI				
10 101	LDD	CPD	IND	OUTD				
10 110	LDIR	CPIR	INIR	OTIR				
10 111	LDDR	CPDR	INDR	OTDR				

Podemos ver inmediatamente lo que hace esta instrucción decodificando sus bits la siguiente forma:

```
0 1 - - - - - - - - - -   instrucción LD
    1 1 1 - - - - -       A es el registro de destino
            0 1 0         D es el registro fuente
```

Así, el código mnemónico para esta instrucción es LD A,D (comprobar en la tabla 6-1) que significa que el Z-80 ponga el contenido del registro D en el acumulador, es decir el registro A es *cargado* con el contenido del registro D.

Las instrucciones aritméticas y lógicas (cuyos códigos de operación empiezan con 10) también ilustran la decodificación de los registros. Por ejemplo, en el grupo

```
      1 0 0 0 0 - - -
al    1 0 1 1 1 - - -
```

los tres últimos bits corresponden al *registro* involucrado. Para las instrucciones siguientes

```
de    0 0 0 0 0 1 0 0
a     0 0 1 1 1 1 0 0

y     0 0 0 0 0 1 0 1
a     0 0 1 1 1 1 0 1

y     0 0 0 0 0 1 1 1
a     0 0 1 1 1 1 1 1,
```

el tercero, cuarto y quinto bits (contando desde la izquierda a derecha) representan el registro que está involucrado en la operación.

Hasta aquí, hemos discutido las instrucciones con un código de operación de un byte. Las tablas 6-2 a 6-5 muestran las instrucciones con códigos de operación multibyte. Ninguno de los códigos operación de varios bytes está implementado en el conjunto de instrucciones del microprocesador Intel 8080A. El conjunto completo de instrucciones comprende solamente códigos de operación de un byte, y excepto para los 8 bytes del recuadro, aparece en la tabla 6-1. Así, la mayor parte de las instrucciones nuevas para el Z-80 aparecen con un código de operación de varios bytes en las tablas 6-2 hasta 6-5.

Los códigos de operación de varios bytes pueden ser divididos en cuatro grupos basándonos en su primer byte: CB, DD, ED o FD. Destacaremos algunas de las propiedades más importantes para cada uno de estos grupos.

Las instrucciones CB (tabla 6-2)

Los códigos de operación de dos bytes que tienen el primer byte igual a CB se

muestran en la tabla 6-2. Los ocho bits que se obtienen a partir de su posición en la columna, para una determinada instrucción representan el segundo byte del código de operación. Por ejemplo, la instrucción BIT 2,C corresponde al siguiente código de operación de dos bytes.

Byte 1 = CB
Byte 2 = 51 (0 1 0 1 0 0 0 1)

La estructura del byte 2 es fácilmente deducible de la tabla 6-2.

Primeros dos bits =

0 0_ _ _ _ _una instrucción de rotación o desplazamiento en la cual los últimos tres bits son el código para el registro involucrado.
0 1_ _ _ _ _una instrucción BIT en la cual los últimos tres bits especifican el registro y los otros tres bits especifican el bit a direccionar.
1 0_ _ _ _ _una instrucción RES en la cual los últimos seis bits tienen el mismo significado que en la instrucción BIT.
1 1_ _ _ _ _una instrucción SET en la cual los últimos seis bits tienen el mismo significado que en la instrucción BIT.

Las instrucciones DD y FD (tablas 6-3 y 6-4)

Las instrucciones que empiezan con DD o FD tienen un código de operación de 2 o 3 bytes. La tabla 6-3 muestra las instrucciones con un código de operación DD de dos bytes. Nótese que todas ellas utilizan el registro índice IX. Los códigos de operación de dos bytes FD están definidos en forma análoga para el registro índice IY.

Todos los códigos de operación de tres bytes implementados en el Z-80 tienen la siguiente estructura:

Byte 1: DD o FD dependiendo del registro índice (IX o IY)
Byte 2: CB
Byte 3: byte de desplazamiento
Byte 4: los primeros dos bytes indican rotación o desplazamiento, BIT, RES, o SET como en las instrucciones CB anteriores. Los próximos tres bytes son siempre 110. Los bits restantes indican el tipo de rotación o desplazamiento o el número del bit como en las instrucciones CB anteriores.

Los códigos de operación de tres bytes DD se muestran en la tabla 6-4 estando determinado el cuarto byte por la posición de la instrucción con respecto a la fila y columna.

Las instrucciones ED (tabla 6-5)

Las instrucciones que empiezan con ED tienen todas códigos de operación de dos bytes. La tabla 6-5 muestra estas instrucciones, estando el segundo byte por la posición de la instrucción en la fila y columna. Los conjuntos que se producen se

dejan como ejercicios para los lectores que están interesados en la decodificación de las instrucciones.

MODOS DE DIRECCIONAMIENTO DEL Z-80

Casi todas las instrucciones del Z-80 realizan operaciones en los datos que están almacenados en los registros del chip de la CPU, en la memoria, o pueden ser entrados o sacados desde las puertas de I/O (entrada/salida). El término *modos de direccionamiento* se refiere al método por el cual la instrucción accede al dato. ¿Forma parte el dato de la instrucción? ¿Contiene la instrucción un código que le dice al computador donde está el dato? ¿Contiene la instrucción un indicador que señala a la posición de memoria donde está guardado el dato? En total, el Z-80 tiene diez modos distintos de direccionamiento. Investigaremos éstos en detalle en éste y en los capítulos subsiguientes.

La variedad y potencia de los modos de direccionamiento del Z-80 contribuyen en gran medida a las muchas ventajas que el chip tiene con respecto a otros microprocesadores de 8 bits, como el Intel 8080A. Desafortunadamente, también se suman a la complejidad del conjunto de instrucciones del Z-80. Sin embargo, esté seguro que el tiempo extra y el esfuerzo desplegado para aprender los modos de direccionamiento serán pagados ampliamente de muchas maneras. Primero, las muchas formas en que los datos se pueden llamar y manipular, lo fácil que resulta escribir programas eficientes. Segundo, Zilog Corporation ha diseñado un método excelente para encontrar los mnemónicos del Z-80 y su asociado código hexadecimal que requiere que el usuario conozca y entienda los diez modos de direccionamiento. Los que perseveren en aprender los modos de direccionamiento se verán recompensados con creces, son demasiado numerosos para tratar de enumerarlos aquí. Una palabra final de advertencia antes de profundizar en el estudio del primer conjunto de modos de direccionamiento: dedique sus esfuerzos iniciales para comprender que *significan* los modos de direccionamiento. La memorización de los nombres puede llegar más tarde con la experiencia. Pensamos que los modos de direccionamiento son importantes, y le instamos a que pase el tiempo necesario para aprenderlos.

INSTRUCCIONES DE CARGA DE UN SOLO REGISTRO: MODO DE DIRECCIONAMIENTO DE REGISTROS LD d,s

Existen 63 instrucciones LD diferentes de un solo registro en el conjunto de instrucciones del Z-80. Todas las instrucciones tienen el código mnemónico LD d,s en

donde

d = registro de destino
s = registro fuente (source)

La gama de los códigos de instrucción va desde 40 a 7F, con la sola excepción de 76 que es la instrucción HALT. La forma de los ocho bits de la instrucción LD es,

0 1 D D D S S S

Los valores DDD o SSS son los tres bits que corresponden al código binario de tres bits específico de un determinado registro. Así:

Registro	DDD o SSS código binario
B	000
C	001
D	010
E	011
H	100
L	101
(HL)	110
A	111

Ahora estamos preparados para definir el modo de direccionamiento por registro:

direccionamiento por registro: La técnica de utilizar grupos de bits en el código de instrucción del Z-80 para especificar qué registro(s) está implicado.

A continuación se resumen algunos ejemplos de la utilización de esta clase de instrucciones.

Cargar el registro C desde el registro B	LD C,B
Cargar el acumulador desde el registro C	LD A,C
Cargar el registro D desde el registro E	LD D,E
Cargar el acumulador desde el registro H	LD A,H
Cargar el registro L desde el acumulador	LD L,A
Cargar la posición de memoria direccionada por el par de registros H y L con el acumulador	LD (HL),A
Cargar el acumulador desde la posición de memoria direccionada por el par de registros H y L	LD A,(HL)

La transferencia de datos entre una posición de memoria, (HL), y cualquier otro registro necesita una explicación adicional, y será discutida en una sección subsiguiente. Las instrucciones LD A,B; LD B,A; LD A,(HL); y LD (HL),A se muestran en la ilustración de la figura 6-1. La flecha señala en la dirección de la transferencia de datos.

CARGAR INMEDIATO A REGISTRO LD r,<B2>

El término *inmediato* se refiere al modo de direccionamiento por el cual el dato que se va a cargar en el registro r está contenido dentro de la instrucción de varios bytes como el byte número dos, <B2>. En la instrucción cargar-inmediato-al registro, el destino del byte de datos está indicado por los bits marcados "D",

Byte 1 0 0 D D D 1 1 0
Byte 2 byte de dato <B2>

Los valores para DDD son los códigos de los registros de la página precedente. El mnemónico es LD r, <B2>. El número de estados es 7, lo que corresponde a un tiempo de ejecución de 2,8 microsegundos con la excepción de la instrucción LD (HL), <B2>, la cual requiere 10 estados o 4 microsegundos. En la figura 6-1 mostramos las ocho instrucciones diferentes de carga-inmediato-a registro. El byte <B2> es el segundo byte en la instrucción de dos bytes; esta información se transfiere desde el programa al registro designado.

Observe que *todos los movimientos de registros y datos en el Z-80 y en el Nanocomputador se hacen en paralelo; los ocho bits de información se transfieren al mismo tiempo.* Existen condiciones especiales en la transferencia de datos a y desde la posición de memoria (HL). Este tópico se discute a continuación.

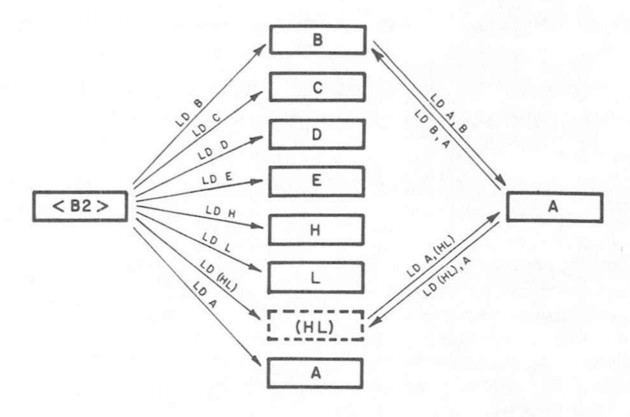


Figura 6-1.

CARGA INDIRECTA DE REGISTRO CON EL ACUMULADOR LD A,(rp); LD (rp),A

Registro indirecto es un modo de direccionamiento en el cual se utiliza un par

de registros para señalar a una dirección de memoria cuyo contenido es o bien reemplazado con, o cargado en, el acumulador A. Por ejemplo LD A,(DE) coloca el byte de ocho bits, cuya dirección de memoria está contenida en el par de registros DE, en el acumulador. LD (DE),A guarda el contenido del acumulador en la posición de memoria que está direccionada por el contenido del par de registros DE.

Los códigos de operación para estas instrucciones son

LD A,(rp):	0 0 r p 1 0 1 0	o	LD A,(HL)	0 1 1 1 1 1 1 0
LD (rp),A:	0 0 r p 0 0 1 0	o	LD (HL),A	0 1 1 1 0 1 1 1

en donde el par de registros se codifica de la siguiente forma. La figura 6-2 muestra como funciona el direccionamiento por registro indirecto.

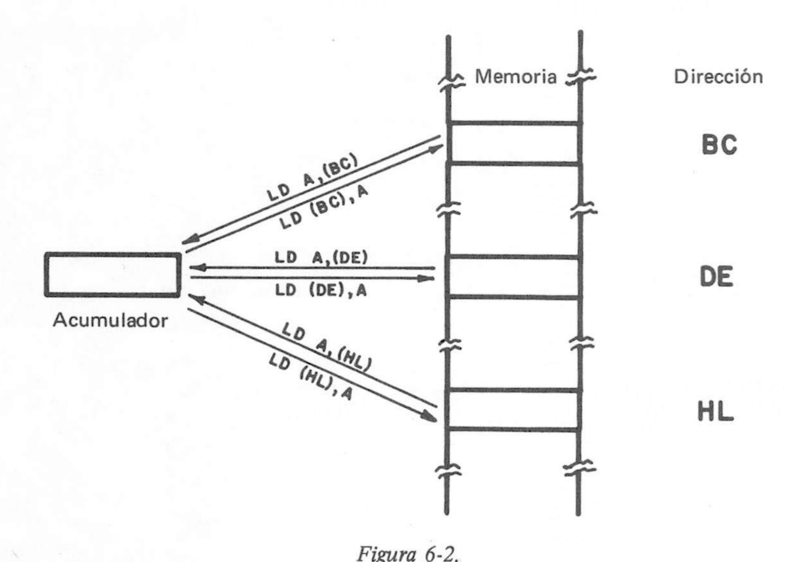


Figura 6-2.

CARGA INMEDIATA EXTENDIDA A UN PAR DE REGISTROS LD rp, <B3> <B2>

Las instrucciones de carga inmediata extendida pertenecen al grupo llamado de "carga de 16 bits" puesto que estas instrucciones provocan la transferencia de dos bytes desde la instrucción (bytes dos y tres) dentro del par de registros: BC, DE, HL o SP. El término *inmediato extendido* se refiere a otra forma de direccionamiento del Z-80 en el cual *extendido* significa una transferencia de dos bytes, e *inmediato* significa que los dos bytes de datos forman parte de la instrucción. La

figura 6-3 muestra varias instrucciones de carga inmediata extendida y una representación gráfica de sus propiedades de transferencia de datos.

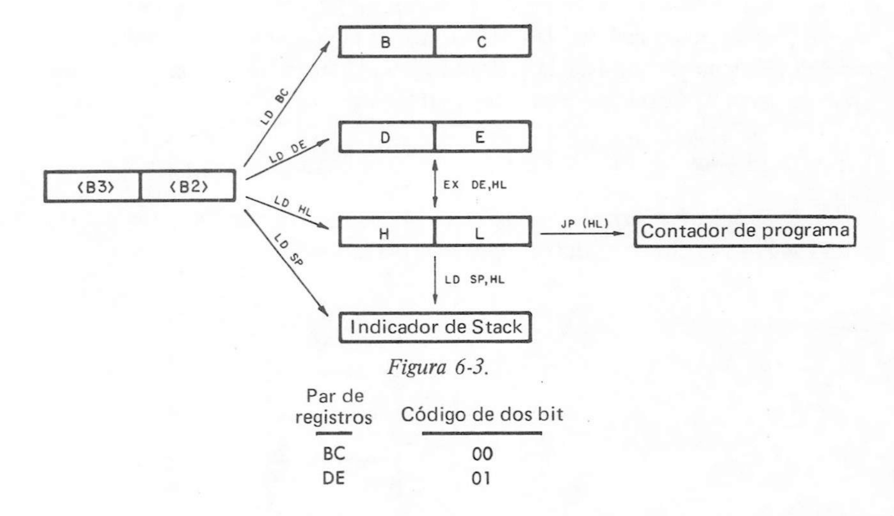


Figura 6-3.

Par de registros	Código de dos bit
BC	00
DE	01

El par de registros de destino de los bytes de datos está indicado mediante los bits marcados "rp" en el código de operación de estas instrucciones:

0 0 r p 0 0 0 1

en donde existe la siguiente correspondencia entre los pares de registros y los códigos de dos bits:

Par de registros	Código binario
BC	00
DE	01
HL	10
SP	11

CARGA EXTENDIDA DE UN PAR DE REGISTROS
LD rp,(direc); LD (direc),rp

Estas instrucciones utilizan *direccionamiento extendido* para mover dos bytes entre la memoria y un par de registros BC, DE o HL. Direccionamiento extendido significa que los dos bytes contenidos en la instrucción señalan al primero de dos bytes situados en la memoria los cuales han de ser la fuente o el destino de la transferencia. Por ejemplo, la instrucción

LD HL, (0100H)

carga el registro L con el dato almacenado en la posición 0100 y el registro H con el dato almacenado en la posición 0101. Nótese que el *segundo* registro en el par se carga con el dato desde la *menor* de las dos direcciones. El código de operación de esta instrucción es 2A así LD HL,(0100H) se traduce a

2A	código de operación
00	byte de dirección de memoria baja (LO)
01	byte de dirección de memoria alta (HI)

Las operaciones análogas de carga para los pares de registros BC y DE tienen códigos de operación de dos bytes. En lugar de listar las instrucciones y sus códigos de operación como lo hemos hecho anteriormente, las presentaremos más adelante en un formato mucho más estructurado en la tabla del grupo de carga de 16 bits en el próximo capítulo. La idea más importante aquí es el entender el direccionamiento extendido.

INCREMENTAR REGISTRO INC r

Incrementar un registro significa aumentar el contenido del registro de 1. Esta instrucción de un solo byte es simple

0 0 r 1 0 0

y tiene el mnemónico INC r. Con la excepción de INC (HL) las instrucciones de incrementar registros necesitan solamente cinco estados, o 2 microsegundos para su ejecución.

DECREMENTAR REGISTRO DEC r

Decrementar un registro significa disminuir el contenido del registro de una unidad. Esta instrucción de un solo byte es similar a la instrucción de incrementar que hemos descrito

0 0 r 1 0 1

y tiene como mnemónico DEC r, en donde r es la identificación del registro (A, B, C, etc.) La instrucción de decrementar registro necesita cinco estados, o 2,0 microsegundos para su ejecución. Ambas instrucciones de incrementar y decrementar utilizan el código binario de tres bits para el registro.

SALTO SI NO CERO JP NZ, <B3><B2>

Esta es su primera *instrucción condicional de salto,* que es una instrucción

que está sujeta a una condición. En este caso el salto se produce a la dirección dada en el segundo <B2> y tercer <B3> bytes de la instrucción *si el indicador de cero está al valor 0 lógico.* No estamos preparados en este momento para hablar acerca de los indicadores de estado; para nuestro propósito el salto ocurre solamente si el resultado de una operación en un registro no es cero. Si el resultado de la operación en el registro es cero, entonces la instrucción JP NZ es ignorada y el programa salta por encima de los tres bytes de la instrucción a la siguiente instrucción. La instrucción JP NZ es una instrucción de tres bytes

1 1 0 0 0 0 1 0
byte de dirección baja (LO) <B2>
byte de dirección alta (HI) <B3>

que tiene un tiempo de ejecución de 10 estados, o 4 microsegundos. Esta instrucción es muy utilizada para la creación de bucles de retardo, un ejemplo del cual se dará en este capítulo con un programa.

TRANSFERENCIA DE BLOQUES DE DATOS LDD, LDI, LDDR, LDIR

Hasta aquí, hemos discutido muchas maneras de transferir datos entre los registros y posiciones de memoria de un byte a la vez. El Z-80 tiene cuatro instrucciones muy poderosas diseñadas para facilitar el movimiento de bloques de datos desde un grupo de posiciones de la memoria a otra. Antes de ejecutar cualquiera de estas cuatro instrucciones, el programa del Z-80 debe *inicializar* los registros BC, DE y HL de la siguiente forma:

HL = dirección del primer byte de origen
DE = dirección del primer byte de destino
BC = número de bytes a mover

La ejecución de la instrucción LDI (*cargar-incrementar*) provoca que ocurran los siguientes pasos:

1. El byte que está en la posición de memoria direccionada por el par de registros H y L es cargado en la posición direccionada por el par de registros DE.
2. El contenido de los pares de registros HL y DE son ambos incrementados (de 1).
3. El contenido del par de registros BC se decrementa (de 1).

La ejecución de la instrucción LDIR (*cargar-incrementar-repetir*) hace que ocurra lo siguiente:

1. El byte situado en la posición de memoria indicada por el par de registros HL es cargado en la posición direccionada por el par de registros DE.
2. El contenido de los pares de registros HL y DE se ve incrementado en ambos pares de registros.
3. El contenido del par de registros BC es decrementado.
4. Se comprueba el valor del par de registros BC. Si BC no es igual a 0000 entonces se repiten los pasos 1, 2, 3 y 4. Si BC es igual a 0000 entonces la ejecución continúa con la próxima instrucción en el programa.

La ejecución de las instrucciones LDD (*cargar-decrementar*) y LDDR (*cargar-decrementar-repetir*) tienen una secuencia muy similar de pasos. La única diferencia es que el Paso 2 *decrementa* a ambos registros HL y DE.

La figura 6-4 muestra los registros y las posiciones de memoria antes y después de la ejecución de la instrucción LDIR. Esta instrucción es el sujeto de un experimento que usted realizará al final de este capítulo.

INTRODUCCION A LOS EXPERIMENTOS

Los siguientes experimentos están diseñados para demostrar lo que ha aprendido en el capıtulo 6 referente a la transferencia de datos entre registros, entre los registros y posiciones de memoria, y entre posiciones de memoria y otras posiciones de memoria.

Los experimentos que usted realizará se pueden resumir de la siguiente forma:

Experimento N.°	Comentarios
1	Demuestra los modos de direccionamiento inmediato y mediante registro.
2	Demuestra los modos de direccionamiento inmediato extendido, extendido y registro indirecto.
3	Demuestra las técnicas para implementar bucles de programa. Específicamente se utiliza la instrucción JP NZ para realizar un bucle de retardo.
4	Demuestra la instrucción LDRR para mover bloques.
5	Demuestra las instrucciones LDI para mover bloques. También se introducen dos nuevos saltos condicionales así como instrucciones lógicas.

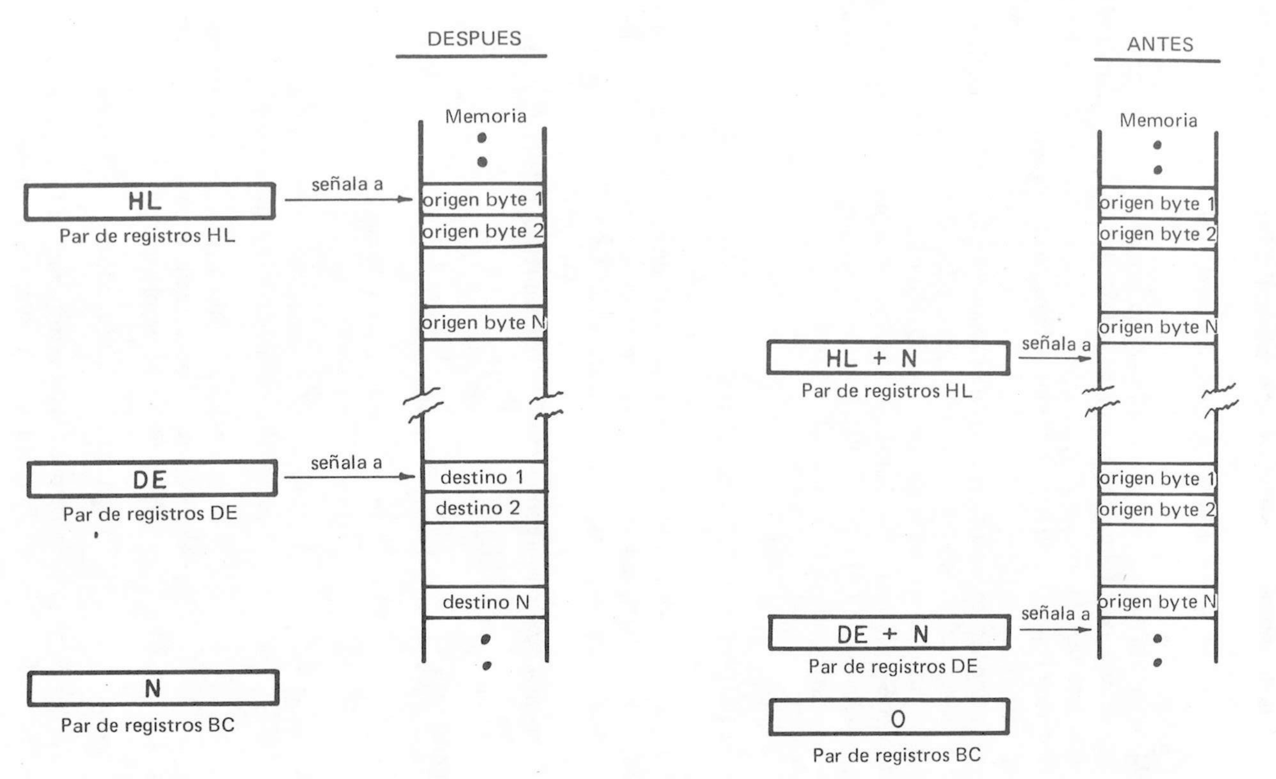

DESPUES
Memoria
señala a
HL
Par de registros HL
origen byte 1
origen byte 2
origen byte N
señala a
DE
Par de registros DE
destino 1
destino 2
destino N
N
Par de registros BC
ANTES
Memoria
origen byte 1
origen byte 2
origen byte N
señala a
HL + N
Par de registros HL
origen byte 1
origen byte 2
origen byte N
señala a
DE + N
Par de registros DE
O
Par de registros BC

Figura 6-4.

6 Demuestra el valor de las instrucciones de movimiento de bloques mostrando como pueden ahorrar memoria y pasos de programa.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar los modos de direccionamiento inmediato y por registro.

Programa N.° 7

Posición de memoria	Código objeto	Código fuente	Comentarios
0100	06 80	LD B,80H	; Direccionamiento directo. El byte de dato ; 80 es cargado en el registro B
0102	04	INC B	; Sumar 1 (para incrementar) al registro B
0103	48	LD C,B	; Direccionamiento por registro el contenido ; del registro B es cargado en el ; registro C
0104	0C	INC C	; Incrementar el contenido del ; registro C
0105	51	LD D,C	; Cargar D con C
0106	14	INC D	; Incrementar D
0107	5A	LD E,D	; Cargar E con D
0108	1D	DEC E	; Disminuir el contenido del registro E ; de 1 (decrementar)
0109	63	LD H,E	; Cargar H con E
010A	25	DEC H	; Decrementar H
010B	6C	LD L,H	; Cargar L con H
010C	2D	DEC L	; Decrementar L
010D	7D	LD A,L	; Cargar A con L
010E	76	HALT	; Parar el microcomputador

Paso 1

Cargar el programa precedente en la memoria empezando en la posición 0100. Verificar que ha sido cargado correctamente.

¿Cuántas instrucciones de un byte hay en el programa anterior?
¿Cuántas instrucciones de dos bytes?
¿Cuántas instrucciones de tres bytes?
¿Cuántas instrucciones de cuatro bytes?

Sus respuestas deberían haber sido 13, 1, 0 y 0, respectivamente. La única instrucción de dos bytes es LD B,80H, mientras que todas las demás instrucciones son de un byte.

¿Cuántas instrucciones utilizan direccionamiento inmediato?
¿Cuántas instrucciones utilizan direccionamiento por registro?

Sus respuestas deberían haber sido 1 y 12 respectivamente. La instrucción LD B,80H utiliza direccionamiento inmediato puesto que el dato forma parte de la misma instrucción (byte dos). Las otras instrucciones LD, INC y DEC (todas con una longitud de un byte) utilizan direccionamiento por registro.

Paso 2

Analice el programa y señale el valor que habrá en cada registro al finalizar la ejecución. Usted puede escribir sus predicciones en el espacio proporcionado.

B=______ C=______ D=______ E=______
H=______ L=______ A=______

Paso 3

Verifique sus predicciones ejecutando el programa. Si usted lo ejecuta en paso a paso, usted puede comprobar su predicción para cada registro en el momento en que es cambiado. Si usted ejecuta el programa a toda velocidad, recuerde de pulsar BREAK en lugar de RESET, de forma que se salve el contenido de los registros.

Nota: Es particularmente interesante observar la ejecución en el modo paso a paso debido a que se puede observar más de un registro a la vez. Con la lámpara de selección en la posición BC usted puede observar el efecto de las cuatro primeras instrucciones, mueva a continuación la lámpara de selección a la posición DE para las próximas cuatro instrucciones, a la posición HL para las siguientes cuatro instrucciones, y a la posición AF para las instrucciones finales.

En los registros se debe leer ahora lo siguiente:

B=81, C=82, D=83, E=82, H=81, L=80, y A=80.

Es esencial ser capaz de predecir el contenido de los registros que se ven afectados por el programa, puesto que usted puede detectar errores en los programas si las predicciones no concuerdan con los resultados. Le recomendamos que trabaje duramente para desarrollar esta habilidad.

Paso 4

Cambie el byte de datos en la posición 0101 a 01. Predecir lo que contendrán los registros B, C, D, E, H, L y A después de ejecutar el programa cambiado.

B=______ C=______ D=______ E=______
H=______ L=______ A=______

Ejecute el programa en modo paso a paso, observando los registros a medida que van cambiando. Después de la ejecución se debe leer en los registros:

B=02, C=03, D=04, E=03, H=02, L=01, y A=01.

Paso 5

Cambiar el byte de datos en la posición 0101 por FF. Predecir lo que contendrán ahora los registros B, C, D, E, H, L y A. Compruebe sus predicciones ejecutando el programa en modo paso a paso, y observando los registros a medida que van cambiando.

B=______ C=______ D=______ E=______
H=______ L=______ A=______

Observamos que B=00, C=01, D=02, E=01, H=00, L=FF, y A=FF. Estos resultados se explican fácilmente si usted conoce un hecho: en el Z-80 la suma es cíclica. Esto es,

Si incrementamos FF de 1 obtenemos 00
Si decrementamos 00 de 1 obtenemos FF

Otra forma de decir esto es que el Z-80 suma en módulo 256 (base 10) o módulo 100 (base 16). Siempre que esto ocurra, es decir que se detecta el "paso por cero"; este acontecimiento es anotado por el Z-80 colocando a 1 el indicador de CARRY (arrastre). Discutiremos esto en un capítulo posterior.

EXPERIMENTO N.° 2

Propósito

El propósito de este experimento es demostrar los modos de direccionamiento inmediato extendido, extendido y registro indirecto.

Programa N.° 8

Posición de memoria	Código objeto	Código fuente	Comentarios
0110	21 1C 01	LD HL,011CH	; Direccionamiento inmediato extendido:
			; H se carga con 01 (HI)
			; L se carga con 1C (LO)
0113	36 FF	LD (HL),FFH	; Direccionamiento registro indirecto.
			; La posición de memoria que señalan
			; el contenido de HL
			; es cargada con FF
0115	2C	INC L	; Incrementar el registro L de forma que
			; HL está señalando a la próxima
			; posición de memoria en secuencia
0116	36 EE	LD (HL),EEH	; Direccionamiento registro indirecto.
			; La posición de memoria señalada
			; por el contenido de HL
			; se carga con EE
0118	2A 1C 01	LD HL,(011CH)	; Direccionamiento extendido: El registro
			; L se carga con el contenido
			; de la posición de memoria 011C y
			; el registro H es cargado con el
			; contenido de la posición de memoria
			; 011D.
011B	76	HALT	; Paro

Paso 1

Cargar el programa anterior en la memoria empezando en la posición 0110. Verificar que el programa se ha cargado correctamente.

Paso 2

Vamos a examinar más de cerca el programa anterior para tratar de entender qué es lo que hace. En primer lugar observe que el programa incluye los registros H y L y las dos posiciones de memoria 011C y 011D. El programa básicamente carga estas dos posiciones con FF y EE, respectivamente, y a continuación, mueve el contenido de estas dos posiciones al par de registro HL. Este no es un programa particularmente excitante, pero ilustra varios hechos importantes acerca de tres clases de direccionamiento.

Considere las dos instrucciones mnemónicas:

```
LD HL,011CH
LD HL,(011CH)
```

La única diferencia es que la segunda instrucción tiene paréntesis rodeando la

dirección. Esta diferencia es crítica. En la primera instrucción 011C representa *dos bytes de datos* que han de ser cargados en HL; en la segunda instrucción 011C es una *dirección.* Ambas instrucciones cargan 16 bits (o dos bytes) de datos pero la primera utiliza direccionamiento *inmediato extendido* y la segunda utiliza direccionamiento *extendido.*

Las instrucciones LD (HL),FFH y LD (HL),EEH sirven ambas para cargar 8 bits puesto que solamente se ve afectado un byte. Los paréntesis alrededor de HL también aquí son críticos e importantes. Implican que las instrucciones no cambian HL sino, por el contrario, cambian las posiciones de memoria señaladas por HL.

Paso 3

Predecir los valores en los siguientes registros y posiciones de memoria después de ejecutar el programa precedente:

H=______ (011C)=______
L=______ (011D)=______

Paso 4

Ejecute el programa a toda velocidad y examine los registros y la memoria para ver si sus predicciones fueron correctas. Observemos H=EE, L=FF, (011C)=FF, (011D)=EE.

EXPERIMENTO N.° 3

Propósito

El propósito de este experimento es el de demostrar las técnicas para implementar un bucle de programa. Se utilizará en particular la instrucción JP NZ para formar un bucle de retardo de tiempo.

Programa N.° 9

Posición de memoria	Código objeto		Código fuente	Comentarios
0120	0E 00		LD C,00H	; Cargar C con el dato inmediato 00
0122	0D	LOOP:	DEC C	; Decrementar C
0123	C2 22 01		JP NZ, LOOP	; Si C no es cero, ir hacia atrás a BUCLE
0126	FF		RST 38H	; Si C es cero, devolver el control al
				; sistema operativo del Nanocomputador.

Programa N.º 10

Posición de memoria	Código objeto		Código fuente	Comentarios
0130	06 00		LD B, 00H	; Cargar B con 00
0132	0E 00	LOOP1:	LD C, 00H	; Cargar C con 00
0134	0D	LOOP2:	DEC C	; Decrementar C
0135	C2 34 01		JP NZ, LOOP2	; Si C no es cero ir hacia atrás a BUCL2
0138	05		DEC B	; Si C es cero, decrementar B
0139	C2 32 01		JP NZ, LOOP1	; Si B no es cero, ir a BUCL1
013C	FF		RST 38H	; Si B es cero, devolver el control
				; al sistema operativo del Nanocomputador.

Paso 1

Cargar y verificar el programa N.º 9.
Cargar y verificar el programa N.º 10.

Paso 2

Vamos a examinar estos dos programas más de cerca para entender exactamente lo que están haciendo. En primer lugar vamos a concentrarnos en el programa número 9. El programa número 9 es simplemente un bucle de retardo. Obsérvese que este programa utiliza *etiquetas* o nombres simbólicos. Esto es, en el programa fuente, se le han asignado nombres a ciertas posiciones de memoria. En el programa número 9 al código mnemónico DEC C de la posición 0122 se le asigna la etiqueta BUCLE. Usted puede decir que BUCLE es una etiqueta porque está seguido por dos puntos (:) y una instrucción. Más tarde, en la instrucción JP NZ BUCLE se refiere a la etiqueta como un sinónimo de la dirección 0122. Esto es, la instrucción JP NZ, BUCLE es equivalente a la instrucción JP NZ, 0122H y el ensamblador la convierte en el código objeto hex C2 22 01. Es importante para usted que comprenda que las etiquetas, que se utilizan en las definiciones del código fuente, son traducidas al código objeto equivalente en hex de forma que la CPU del Z-80 *NUNCA* ve la etiqueta.

Observe también que el programa número 9 utiliza la instrucción RST 38H que no ha sido definida por el momento. Esta instrucción le dice a la CPU Z-80 que devuelva el control al sistema operativo de Nanocomputador. Más tarde discutiremos la instrucción RST en detalle.

El registro C es cargado inicialmente con ceros, entonces el registro C es decrementado repetidamente hasta que C alcanza de nuevo 00, en cuyo caso finaliza el bucle y el control es devuelto al sistema operativo del Nanocomputador. El programa número 9 se le llama un bucle de retardo debido a que sólo realiza el trabajo de decrementar C durante un cierto tiempo y a continuación se para. El

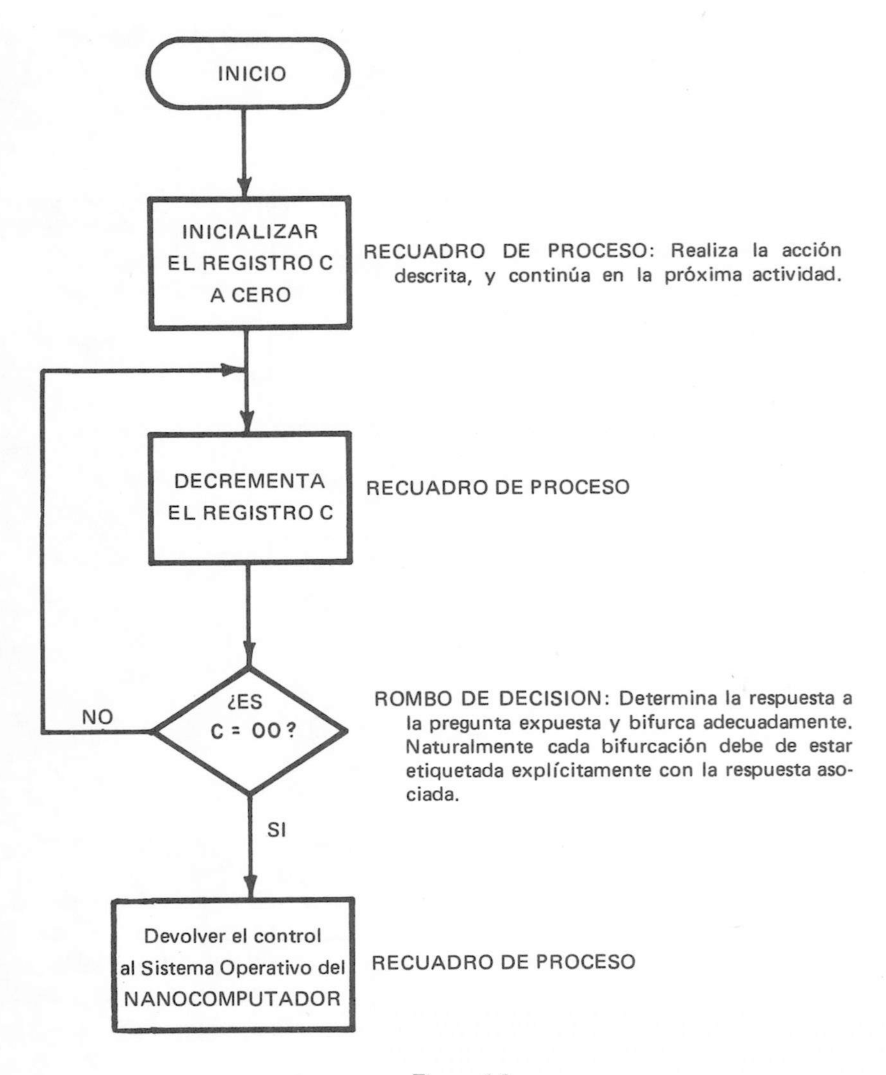


Figura 6-5.

resultado final es un tiempo de retardo, obsérvese que C empieza y termina con ceros.

Para los programas con bucles, a menudo es muy útil dibujar *diagramas de flujo* para ilustrar la lógica completa del programa. El diagrama de flujo para el programa número 9 se da en la figura 6-5. Explicaremos el significado de la forma de las "cajas" en el diagrama de flujo.

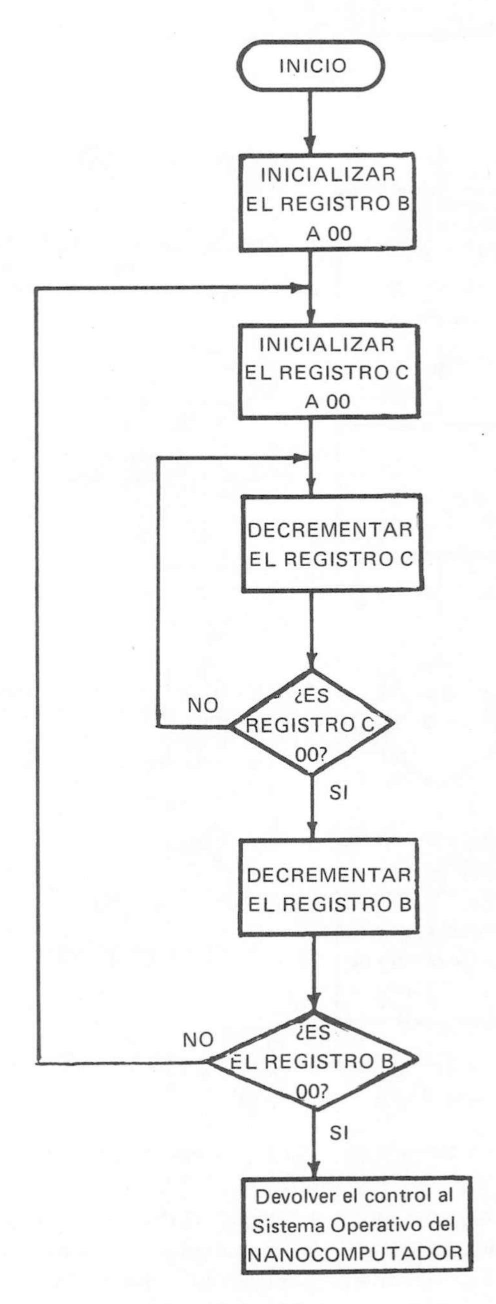

INICIO
INICIALIZAR EL REGISTRO B A 00
INICIALIZAR EL REGISTRO C A 00
DECREMENTAR EL REGISTRO C
NO
¿ES REGISTRO C 00?
SI
DECREMENTAR EL REGISTRO B
NO
¿ES EL REGISTRO B 00?
SI
Devolver el control al Sistema Operativo del NANOCOMPUTADOR

Figura 6-6.

Es apropiado hacer algunas observaciones generales acerca de los bucles de programa. Todos los bucles de programa pueden ser analizados incluyendo los cuatro componentes siguientes:

1. *Proceso de inicialización:* Las variables de contaje, direcciones de memoria, registros y otras variables necesarias se colocan a los valores iniciales deseados (por ej. LD C, 00H en el programa número 9).
2. *Proceso a repetir:* Este componente está constituido de las instrucciones que se ejecutarán en cada bucle. (Nota: Este componente no existe en el programa N.º 9. Este es el motivo por el que se le llama un bucle de retardo de tiempo.)
3. *Proceso de control del bucle:* Las variables de contaje y todos los indicadores de memoria u otros valores que controlan la frecuencia de repetición del bucle se actualizan (por ej. DEC C en el programa N.º 9).
4. *Proceso de final del bucle:* Se hace una comprobación de las variables de control del bucle para determinar si se ha alcanzado la condición de finalizar. Se continúa el proceso o se para de acuerdo con este resultado (por ej. JP NZ, BUCLE en el programa número 9).

Paso 3

Vamos ahora a analizar el programa número 10 de una forma similar. En primer lugar, intente dibujar un diagrama de flujo para el programa N.º 10. Compare el suyo con el que aparece en la figura 6-6.

Obsérvese que a partir del diagrama de flujo, se muestra claramente la estructura del programa número 10: Este programa consta de un bucle dentro de otro bucle. Para cada vez que el registro B se decrementa una vez, el registro C se decrementa desde 00 a 00 de nuevo es decir ¡256 veces! Por favor, asegúrese de que usted entiende lo que está sucediendo aquí. Cuando el registro C se ha decrementado 256 veces el registro B se ha decrementado una vez. Así, ¿cuál de los dos programas piensa usted que forma un bucle de retardo . . ., el número 9 o el número 10?

Esperamos que usted haya dicho ¡el número 10!

Ejecute el programa N.º 9 a toda velocidad. ¿Qué es lo que observa? y ¿cómo se compara su comportamiento con el del programa N.º 9?

En ambos casos, observamos que el display del Nanocomputador queda apagado durante un breve instante y a continuación todos los dígitos reaparecen con la lámpara de selección en la posición PC. Para el programa número 9, el tiempo que transcurre entre que se pulsa GO y que el display se ilumina era instantáneo, mientras que el lapso de tiempo entre los mismos acontecimientos para el programa número 10 fue mayor. Tal vez medio segundo.

Paso 4

Vamos a investigar como podemos hacer mayor el tiempo de retardo desde que se pulsa GO y que el display se ilumina. Una forma consiste en añadir otro bucle, haciendo tres bucles anidados. En el programa N.° 10, el bucle que decrementa el registro C está *anidado* dentro del bucle que decrementa el registro B.

Cambie el programa número 10 de la siguiente forma:

Posición de memoria	Código objeto	Código fuente	Comentarios
012E	16 30	LD D, 30H	; Inicializar el contador más ; exterior
---- guardar las posiciones 0130-013B sin variación ---- estas instrucciones forman el ------ bucle interno en este momento ----			
013C	15	DEC D	; Decrementar el contador más ; externo
013D	C2 30 01	JP NZ, 0130H	; Volver a repetir los dos bucles internos ; si D no es cero
0140	FF	RST 38H	; Si D es 0, devolver el control al sistema ; operativo del Nanocomputador.

Paso 5

Dibujar un diagrama de flujo del programa N.° 10 cambiado, cuya dirección de inicio es 012E. Estudiarlo a fondo para entender las funciones de cada uno de los tres bucles.

Paso 6

Ejecutar el programa empezando en 012E. Usted debe esperar un retardo mucho más largo desde que se pulsa GO y el control retorna al sistema operativo del Nanocomputador (se vuelven a encender los displays). Esperar pacientemente, el retardo es mucho mayor. (Si no ha sucedido nada después de un minuto, algo va mal. Pulse RESET y compruebe nuevamente que su programa esté cargado correctamente.)

Paso 7

Usted puede cambiar el tiempo de duración del bucle de retardo variando el valor inicial del registro D. Cuanto mayor sea el valor, mayor será el tiempo de retardo. Pruebe distintos valores para el byte de datos situado en la posición 012F para verificarlo.

EXPERIMENTO N.° 4

Propósito

El propósito de este experimento es el de demostrar la instrucción de movimiento de bloques LDDR.

Programa N.° 11

Posición de memoria	Código objeto	Código fuente	Comentarios
0150	21 75 01	LD HL,0175H	; Especificar la dirección final
			; del bloque fuente de datos
0153	11 6F 01	LD DE,016FH	; Especificar la dirección final
			; de destino
0156	01 05 00	LD BC,0005H	; Especificar el número de bytes
			; a transferir
0159	ED B8	LDDR	; Mover el bloque de bytes completo
01FB	FF	RST 38H	; Transferir el control al sistema
			; operativo del Nanocomputador.

Paso 1

Cargar el programa en la memoria empezando en 0150. Verificar que se ha cargado correctamente.

Paso 2

Examine el programa atentamente para descubrir exactamente que es lo que hace. Básicamente, el programa utiliza la instrucción LDDR para mover cinco bytes de datos. La instrucción LDDR es una de las muchas nuevas y potentes instrucciones del Z-80 que no están implementadas en el antiguo microprocesador Intel 8080. El siguiente diagrama ilustra como actúa la instrucción LDDR en el programa precedente:

```
LDDR: (0175) transferido a (016F)
      (0174) transferido a (016E)
      (0173) transferido a (016D)
      (0172) transferido a (016C)
      (0171) transferido a (016B)
```

Se han movido un total de BC=0005 bytes en el orden en el cual aparecen en el diagrama.

Paso 3

Inicialice los contenidos de las posiciones de memoria 0171-0175 de la siguiente forma

(0171) = AA
(0172) = BB
(0173) = CC
(0174) = DD
(0175) = FF

Ejecute el programa a toda velocidad; y examine las posiciones de memoria 016B-016F. Escriba sus observaciones:

(016B) =
(016C) =
(016D) =
(016E) =
(016F) =

Observamos AA, BB, CC, DD y EE respectivamente, en las posiciones precedentes. Así, la instrucción LDDR hace que se transfieran cinco posiciones de memoria. Si BC se hubiera inicializado a 0006 o 0003, entonces se habrían transferido 6 o 3 bytes. Reinicialice los bytes de memoria 016B, 016F a, digamos 11 . . . y pruebe BC=0003 y BC=0006 cambiando el programa adecuadamente (posiciones 0157 y 0158).

Paso 4

Vamos ahora a poner a cero los doce bytes de memoria desde 016A hasta 0175 inclusive, haciendo algunos cambios en el programa precedente.

1. Cambiar LD DE, 016FH a LD DE, 0174H
2. Cambiar LD BC, 0005H a LD BC, 0014H
3. Guarde 00 en la posición de memoria 0175 utilizando el teclado.

Ejecute el programa empezando en 0150. Examine las posiciones de memoria 016A hasta 0175. ¿Están todas a cero?

Observamos que así es.

Paso 5

Vamos a intentar explicar lo que acabamos de hacer. Primero, aquí está el código fuente para el programa con los cambios precedentes incorporados:

```
LD HL, 0175H
LD DE, 0174H
LD BC, 0014H
LDDR
RST 38H
```

Así, la secuencia de las transferencias es

```
(0175)--(0174)
(0174)--(0173)
(0173)--(0172)
. . .
(0162)--(0161)
```

Se han transferido un total de 20 (base 10) o 14 (base 16) bytes. Al cargar 00 en la posición de memoria 0175 (paso 3 anterior), ha empezado un efecto de dominó. La primera transferencia puso a cero la posición 0174, entonces el contenido de 0174 que era de 00 se transfirió a 0173, y así sucesivamente . . .

Paso 6

Anote cuidadosamente los valores de todos los tres pares de registros después de la ejecución del programa:

	Nuestras observaciones
HL=______	0161
DE=______	0160
BC=______	0000

HL es uno menos que la dirección del último byte fuente transferido. DE es uno menos que la dirección del último byte de destino transferido. BC=0000.

Paso 7

¿Qué sucede si ejecutamos el programa anterior con BC inicializado a 0000? Existen dos posibilidades dependiendo de la forma en que el Z-80 ejecuta una instrucción LDDR. Considere los dos casos siguientes para un Z-80 que acaba de encontrar una instrucción LDDR:

Caso N.° 1:
- Paso 1: transferir el byte de dato: (HL) a (DE)
- Paso 2: decrementar HL, DE y BC
- Paso 3: comprobar si BC=0000. Si no, volver al paso 1, de lo contrario ir a la próxima instrucción.

Caso N.° 2:
- Paso 1: comprobar si BC=0000. Si no, continuar con el paso 2, de lo contrario ir a la próxima instrucción.

Paso 2: transferir el byte de dato: (HL) a (DE)
Paso 3: decrementar HL, DE y BC, volver al paso 1.

Si BC no es inicialmente cero, entonces los dos casos anteriores producen resultados idénticos. ¿Qué pasa si BC no se inicializa a cero? Entonces los dos casos difieren drásticamente. El caso n.º 1 intenta mover 64K bytes mientras que en el caso n.º 2 se moverán cero bytes.

Hagamos un simple test para ver cual de los dos casos sigue el Z-80. Simplemente cambie el programa reemplazando el 14 en la posición 0157 con 00. ¿Qué es lo que hace esto? La instrucción LD BC, 0014 es reemplazada con la instrucción LD BC, 0000.

Paso 8

Ejecute el programa (empezando en 0150) en modo paso a paso. En primer lugar observe como se inicializan los tres pares de registros (3 pasos). Ahora *observe el par de registros BC para el paso 4.* ¿Qué es lo que observa? Esta observación servirá para nuestra pregunta. ¿Qué caso sigue el Z-80, el 1 o el 2?

Observamos que BC se convierte en FFFF. Así, el Z-80 decrementa *antes* de hacer el test, por lo que podemos deducir que se efectúa el caso n.º 1 y no el caso n.º 2. Acabamos de investigar una *condición límite* de la instrucción LDDR. Estas condiciones límites se refieren al control del bucle en la primera y/o última iteración y son siempre extremadamente críticas. Muchos errores en los programas en la ejecución de los bucles son debidos a la implementación incorrecta de las condiciones límites.

Paso 9

Vamos a proseguir nuestras observaciones precedentes un paso más. Acabamos de iniciar el movimiento de un bloque de bytes de 64K de memoria. Pero, lo más importante es que estamos empezando a escribir encima de nuestro programa. Considere las siguientes transferencias que van a tener lugar:

```
(0175)–(0174)
(0174)–(0173)
   . . .
(0160)–(015F)
(015F)–(015E)
   . . .
(015B)–(015A)
(015B)–(0159)
```

Con la transferencia del contenido de la posición de memoria 015E a la posición 015D, estamos empezando a alterar el programa que se está ejecutando actualmente. El Z-80 no se da cuenta de ello por ahora porque simplemente está ejecutando la instrucción LDDR en las posiciones 0159 y 0160.

Continúe la ejecución paso a paso del programa observando el registro DE. Cuando DE=015D usted está empezando a escribir encima de su programa (llamado a veces "destrozar" o "comerse") su propio programa. ¿Hasta dónde cree usted piensa que va a llegar? Continúe avanzando. Con cada paso el registro DE se acerca de un byte a la instrucción LDDR. ¿Cuándo terminará el registro DE de decrementar?

Observamos que termina en 0159. Esto es, cuando se ha cambiado el segundo byte de la instrucción LDDR el programa queda aparte, de forma que trabajará de una forma errática ¡hasta que pueda! Una vez que la instrucción que estaba ejecutando fue destruida, no puede continuar con lo que estaba haciendo.

Usted ha visto, como bajo condiciones muy controladas, un programa puede destruirse a sí mismo. Esto, tristemente, no será la última vez que le suceda. Solamente recuerde que puede suceder y procure que esto no suceda. Siempre que ejecute un programa que usted trata de corregir, prepárese para lo peor copiándolo primero en un cassette (si es posible) o por lo menos hágalo bien documentado, porque puede desaparecer después de que pulse la tecla GO.

EXPERIMENTO N.° 5

Propósito

El propósito de este experimento es el de demostrar la instrucción LDI. Se introducen dos nuevas instrucciones de salto condicional JP Z y JP NZ. También se introduce la instrucción OR A.

Programa N.° 12

Posición de memoria	Código objeto	Código fuente	Comentarios
0180	21 A0 01	LD HL, 01A0H	; Especificar la dirección de principio ; del bloque de datos de origen
0183	11 C0 01	LD DE, 01C0H	; Especificar la dirección de principio ; del bloque de datos de ; destino

```
0186    01 10 00          LD BC, 0010H   ; Especificar el número máximo
                                         ; de bytes a mover
0189    7E         LOOP:  LD A, (HL)     ; Cargar el próximo byte de origen que
                                         ; se va a tranferir en el registro A
018A    B7                OR A           ; Colocar el indicador de cero a 1
                                         ; lógico si A es 0
018B    CA 93 01          JP Z, QUIT     ; Si A es cero, saltar al final del
                                         ; programa
018E    ED A0             LDI            ; Transferir el byte que no es cero
0190    EA 89 01          JP PE, LOOP    ; Saltar hacia atrás para transferir otro
                                         ; byte si BC no es 0000
0193    FF         QUIT:  RST 38H        ; Devolver el control al sistema operativo
                                         ; del Nanocomputador.
```

Paso 1

Cargar el programa precedente empezando en la dirección 0180. Verificar que se ha cargado correctamente.

Paso 2

En primer lugar vamos a describir las nuevas instrucciones que aparecen en este programa:

Código objeto	Código mnemónico	Operación
B7	OR A	Realiza la función lógico 0, bit a bit, del acumulador consigo mismo. El indicador de cero se coloca a 1 si A es cero, de lo contrario el indicador de cero es colocado a 0. Ver el capítulo de las instrucciones lógicas para una descripción más completa de esta instrucción.
CA <B2> <B3>	JP Z, <B3><B2>	Salto condicional hacia atrás a la dirección dada por <B3><B2> si el indicador de cero está al valor lógico 1.
EA <B2> <B3>	JP PE, <B3><B2>	Salto condicional: saltar a la dirección dada por <B3><B2> si el indicador de paridad está al valor lógico 1.

Los dos saltos condicionales que se han descrito son muy similares a la instrucción JP NZ. La única diferencia es que el test de la condición de dar o no el salto, se hace antes para decidir si se ha de saltar o no. Estas condiciones siempre incluyen *flags* (indicadores), los cuales serán discutidos con mayor detalle más adelante. Por ahora, le decimos solamente lo que necesita para entender el programa precedente.

Paso 3

Vamos ahora a examinar el programa, en conjunto, para entender exactamente lo que está haciendo. Las tres primeras instrucciones inicializan los pares de registros HL, DE y BC para prepararse para llamar a la instrucción LDI. Las próximas dos instrucciones están destinadas a determinar si el próximo byte que se va a transferir es 00: el byte (que está indicado mediante HL) es cargado en el acumulador y se le hace la operación lógica O consigo mismo. El único caso en que esta operación lógica O dará como resultado 0 si también A es cero. Así, OR A coloca el indicador de cero al valor lógico 1, solamente si A es cero. La próxima instrucción, el salto condicional JP Z, examina el indicador de cero, si está colocado a 1 lógico entonces A es cero, esto es, el próximo byte a ser transferido es 00, de forma que se ejecuta el salto a QUIT el cual devuelve el control del programa al sistema operativo del Nanocomputador. Por otra parte, el salto condicional a la posición 018B no es ejecutado si el próximo byte que se ha de transferir no es cero. Así, se ejecuta la instrucción LDI (es decir, es transferido el byte y HL, DE son incrementados mientras que BC es decrementado). Un hecho crucial que a menudo es ignorado acerca de las instrucciones LDI y LDD es que mientras que BC NO ES CERO, el indicador de paridad es colocado a 1 lógico. Así, cuando BC es decrementado, se hace un test, y el indicador de paridad se coloca de acuerdo con el mismo. Así, el salto condicional JP PE comprueba el indicador de paridad. Si está a 1 lógico, entonces BC no es cero, de forma que el ciclo se inicia de nuevo determinando si el próximo byte a transferir es cero. Si el indicador de paridad está en el valor lógico 0, se han transferido todos los bytes de forma que el control es devuelto al sistema operativo del Nanocomputador, es decir, no se ejecuta JP PE.

El último párrafo es una descripción muy complicada en lenguaje español de un programa. Usted puede ahora apreciar el dicho "un dibujo vale más que 1K de palabras" cuando contempla el diagrama de flujo de la figura 6-7.

Por ahora, debe quedar claro que el programa transfiere un bloque de memoria de 16 bytes de longitud como máximo. El primer byte que está a cero en el bloque de origen termina la transferencia.

Paso 4

Inicialice el bloque de memoria de 16 bytes empezando en 01A0 con bytes que no sean cero, digamos 11. Ejecutar el programa en modo paso a paso examinando los pares de registros BC, DE y HL y viendo como cambian. ¿Cuáles son los valores finales de estos registros después de que se ha devuelto el control al sistema operativo del Nanocomputador?

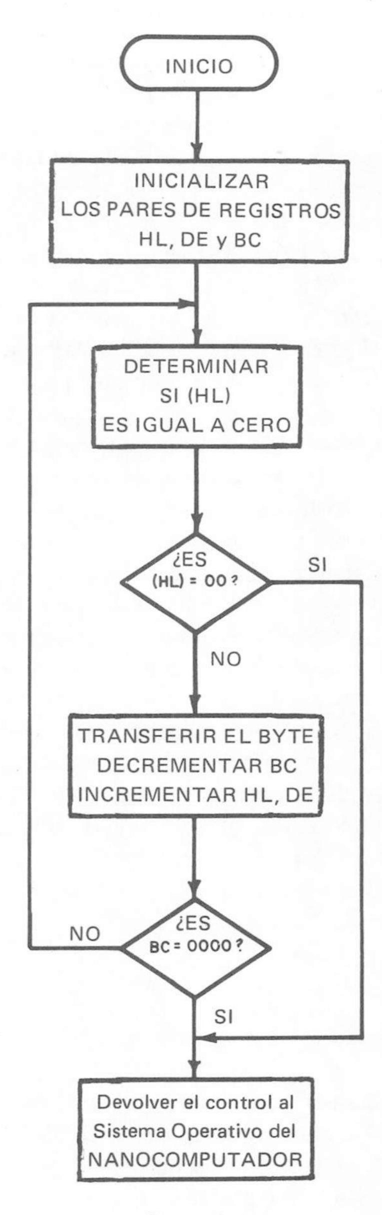

INICIO
INICIALIZAR
LOS PARES DE REGISTROS
HL, DE y BC
DETERMINAR
SI (HL)
ES IGUAL A CERO
¿ES
(HL) = 00 ?
SI
NO
TRANSFERIR EL BYTE
DECREMENTAR BC
INCREMENTAR HL, DE
¿ES
BC = 0000 ?
NO
SI
Devolver el control al
Sistema Operativo del
NANOCOMPUTADOR

Figura 6-7.

HL=
DE=
BC=

Observamos que HL=01B0, DE=01D0, y BC=0000.

Paso 5

Inicialice el bloque de memoria de 16 bits empezando en 01A0 de la siguiente forma:

01A0=10
01A1=0F
01A2=0E
01A3=00
01A4 hasta 01AF=FF

Ejecute de nuevo el programa. Observar el par de registros BC y el registro A. ¿Qué sucede con el registro BC después de que el registro A es cargado con 00? ¿Cuáles son los valores finales de los registros HL, DE y BC?

Usted debe haber observado que el registro BC permanece constante a 000D. Los valores finales de HL y DE son 01A3 y 01C3 respectivamente. Solamente se han transferido tres bytes, como se anticipó.

Obsérvese que el programa no se puede implementar con la instrucción LDIR porque se necesita alguna manipulación de los datos entre las transferencias.

EXPERIMENTO N.° 6

Propósito

El propósito de este experimento es el de demostrar el valor de las instrucciones de movimiento de bloques, mostrando como pueden ahorrar memoria y pasos de programa.

Programa N.° 13: Con LDIR

Posición de memoria	Código objeto	Código fuente	Comentarios
01D0	21 00 02	LD HL, 0200H	; Inicializar los tres pares de registros
01D3	11 01 02	LD DE, 0201H	; de movimiento de bloques para especificar
01D6	01 64 00	LD BC, 0064H	; el origen, destino, y número
			; de bytes
01D9	ED B0	LDIR	; Mover el bloque de datos
01DB	FF	RST 38H	; Transferir el control al sistema operativo
			; del Nanocomputador.

Programa N.° 14: Sin LDIR

Posición de memoria	Código objeto		Código fuente	Comentarios
01D0	21 00 02		LD HL, 0200H	; Lo mismo que antes
01D3	11 01 02		LD DE, 0201H	; Lo mismo que antes
01D6	01 64 00		LD BC, 0064H	; Lo mismo que antes
01D9	7E	LOOP:	LD A, (HL)	; Cargar el byte de origen al Acc.
01DA	12		LD (DE), A	; Guardar en el destino
01DB	23		INC HL	; Actualizar HL: incremento de 16 bits
01DC	13		INC DE	; Actualizar DE: incremento de 16 bits
01DD	0B		DEC BC	; Actualizar BC: decremento de 16 bits
01DE	78		LD A, B	; Test para ver si BC = 0000 - -
01DF	B1		OR C	; Este es un truco que vale la pena
				; recordar. Se discute en detalle en
				; la lección 9
01E0	C2 D9 01		JP NZ, LOOP	; Si BC no es cero, transferir
				; otro byte
01E3	FF		RST 38H	; De lo contrario retornar el control
				; al sistema operativo del Nanocomputador.

Paso 1

Este experimento está diseñado para mostrarle los ahorros increíbles que se pueden obtener con la instrucción LDIR y otras instrucciones de movimiento de bloques, en los programas que mueven datos. Ambos programas mueven un bloque de 100 bytes de memoria consecutivos. Sin embargo, el programa número 13 ocupa 14 bytes de memoria y 2095 estados de CPU o (2095 × 0,000004) = 0,00838 segundos para ser ejecutado. El programa número 14 ocupa 22 bytes de memoria y 5000 estados de CPU (5000 × 0,000004) = 0,020 segundos para ejecutarse . . . ¡más del doble de tiempo! (Discutiremos la metodología para obtener estos tiempos de ejecución con detalle en el apéndice.)

Obsérvese que la discrepancia es más pronunciada a medida que el número de bytes a ser transferidos aumenta. La razón de que esta comparación sea interesante es debido a que el microprocesador 8080 de Intel no tiene instrucciones de movimiento de bloques. Así en un 8080, el procedimiento para mover bloques de datos debe ser el del programa número 14.

Paso 2

Obsérvese que hay varias instrucciones nuevas presentes en el programa N.° 14. Mejor que proporcionar aquí una discusión detallada, diferiremos las explicaciones para más tarde. Aquí nuestro mayor interés consiste en ilustrar la utilidad y eficacia

de las instrucciones LDIR y otras que tratan del movimiento de bloques de información.

Paso 3

Cargar y ejecutar cada uno de los programas precedentes y demostrar que el programa número 13 realiza precisamente la misma función que el programa número 14.

7

Modos de direccionamiento del Z-80

Este capítulo continúa con la descripción de los modos de direccionamiento del Z-80 empezada en el capítulo 6. En particular se investiga la capacidad de direccionamiento indexado que es especialmente importante. Como que se necesitan unos conocimientos de la aritmética en complemento a dos para entender el direccionamiento indexado, hemos introducido una sección en este tópico. Al final de este capítulo introducimos una forma tabular para mostrar los mnemónicos de las instrucciones y sus códigos de operación asociados. Este método fue sugerido primeramente por Zilog Corporation en su Manual Técnico de la CPU del Z-80. Lo hemos encontrado muy útil.

OBJETIVOS

Al finalizar este capítulo, usted será capaz de hacer lo siguiente:

- Definir la representación binaria en complemento a dos para cualquier número.
- Utilizar la aritmética de complemento a dos para realizar operaciones que utilizan direccionamiento indexado.
- ar todos los modos de direccionamiento del Z-80 y dar ejemplos de ciones para cada uno de ellos.
- el stack (pila) y sus operaciones asociadas: PUSH y POP.
- r y utilizar las instrucciones de intercambio.

- Entender y utilizar las tablas de instrucciones de Zilog para los siguientes grupos de instrucciones del Z-80:

  carga de 8 bits
  carga de 16 bits
  transferencia de bloques
  intercambios

¿QUE ES UN MODO DE DIRECCIONAMIENTO?

La noción de modo de direccionamiento fue introducida en el capítulo previo. Por el interés de completarla daremos aquí una definición formal.

modo de direccionamiento: La técnica mediante la cual una instrucción se refiere a los datos con los cuales operará. Las instrucciones del Z-80 implementan un total de diez modos de direccionamiento, con algunas instrucciones que combinan dos modos de direccionamiento para acceder al dato afectado.

En el capítulo 6, describimos los modos de direccionamiento por registro, registro indirecto, inmediato, inmediato extendido, y modo de direccionamiento extendido. Los otros modos de direccionamiento son: página cero modificada, relativo, indexado implicado y direccionamiento de bit. El direccionamiento relativo e indexado proporcionan grandes facilidades para el programador del Z-80, y requieren un buen conocimiento de la aritmética binaria en complemento a dos, que es el sujeto de la próxima sección.

REPRESENTACION BINARIA EN COMPLEMENTO A DOS

En el capítulo 1 definimos un código digital como un sistema de símbolos que representan valores de los datos en una forma útil a los computadores u otros circuitos digitales. La representación en complemento a dos es una forma de codificar números enteros y que es muy similar a la codificación binaria. La diferencia es que los números enteros negativos así como los positivos se pueden codificar utilizando la representación en complemento a dos. Además la representación en complemento a dos hace que la suma y la resta puedan ser implementadas fácilmente mediante circuitos digitales.

En la tabla resumida que sigue, mostramos números decimales positivos y negativos y su representación asociada en 4 bits y complemento a dos. Debemos siempre especificar el número de bits en la representación a complemento a dos (por razones que serán obvias más tarde).

Número decimal	Representación 4 bits en complemento a dos
7	0 1 1 1
6	0 1 1 0
5	0 1 0 1
4	0 1 0 0
3	0 0 1 1
2	0 0 1 0
1	0 0 0 1
0	0 0 0 0
−1	1 1 1 1
−2	1 1 1 0
−3	1 1 0 1
−4	1 1 0 0
−5	1 0 1 1
−6	1 0 1 0
−7	1 0 0 1
−8	1 0 0 0

Podemos hacer varias observaciones:

1. La representación binaria normal con cuatro bits nos permite representar números decimales desde 0 hasta 15. La codificación en complemento a dos con cuatro bits comprende los enteros entre −8 y +7, siendo la mitad de los códigos positivos y la otra mitad negativos. Así la codificación con n bits en complemento a dos codifica los números entre −2**(N−1) y +(2**(N−−1))−1.
2. Los números positivos tienen todos los códigos en complemento a dos con el primer bit igual a cero, mientras que los códigos de los números negativos empiezan con 1. Así, dado un número de cuatro bits en complemento a dos es fácil determinar cuando el número es positivo o negativo. Examine solamente el primer bit (el más significativo). Esto es cierto para los números en complemento a dos de n bits.
3. El código de un número decimal positivo en complemento a dos es idéntico a su código binario.
4. Mientras −8 tiene una representación de cuatro bits en complemento a dos, +8 no la tiene. En la representación de n bit, −2**(N−1) se puede representar pero +2**(N−1) no se puede representar.
5. El "complemento a dos" de 0001 es 1111, de 0101 es 1011, de 1010 es
Esto es, decir que dos números en complemento a dos se "complemen-
o al otro" significa que representan los números decimales negativos
o, o que *sumados dan cero.* ¿La suma resultante es cero? Vamos a

```
  0 0 0 1        1 0 1 0        0 1 0 1
 +1 1 1 1       +0 1 1 0       +1 0 1 1
 ---------      ---------      ---------
 1 0 0 0 0      1 0 0 0 0      1 0 0 0 0
```

Realizando la suma binaria precedente obtenemos algo que no parece cero. Pero recuerde, ¡usted está utilizando solamente representaciones de 4 bits! Así, en el momento en que hacemos la suma del último bit en nuestra suma, ¡hemos sobrepasado el *número de bits*! Así, la respuesta es cero. Este es el porqué debemos siempre especificar el número de bits cuando hablamos de la representación en complemento a dos para números decimales u otros enteros.

La última observación es especialmente importante porque muestra cual es la esencia de la codificación en complemento a dos. Facilita la suma de enteros. También facilita la resta porque restar un número es equivalente a sumarle su complemento a dos. Dado un número binario de n bits, ¿cómo puede uno encontrar su complemento a dos? Se lo mostraremos mediante un ejemplo de 4 bits. Considere el número 0001. Para determinar su complemento a dos, primero cambie todos los números que están en 0 lógico a 1 lógico y todos los bits que estén en 1 lógico a 0 lógico (el resultado para este ejemplo es 1110); a continuación sume 0001 para obtener 1111. Comprobar en la tabla para ver si esto es correcto. Aquí hay varios ejemplos.

Ejemplo 1

Encontrar el complemento a dos de 1010.

```
    Paso 1:   0101
    Paso 2:  +0001
             -----
Respuesta:    0110
```

Ejemplo 2

Encontrar el complemento a dos de 0000.

```
    Paso 1:   1111
    Paso 2:  +0001
             -----
Respuesta:    0000
```

Puesto que –0=0, esto ¡no es sorprendente!

Ejemplo 3

Encontrar el complemento a dos de 1000.

Paso 1:	0111
Paso 2:	+0001
Respuesta:	1000

Observe que el primer bit es ¡un uno lógico!

¡CUIDADO! El complemento a dos de un número negativo debe ser positivo. Hemos indicado porque decimos que –8 (cuya representación en complemento a dos es 1000) no tiene complemento a dos. La razón para esto es que su complemento a dos es +8 el cual no tiene una representación en complemento a dos de 4 bits. Todos los otros números en complemento a dos de 4 bits (entre –7 y +7) tienen complementos de 4 bits.

Vamos a examinar algunas representaciones en complemento a dos de 8 bits.

Ejemplo 4

¿Cuál es el mayor número positivo entero que se puede representar con un código en complemento a dos de 8 bits?

Respuesta: Un número positivo debe empezar con 0, así, el mayor número entero es 0 1 1 1 1 1 1 1 = 127 (base 10)

¿Cuál es el mayor número negativo (mayor en valor absoluto) que se puede representar con un código en complemento a dos de 8 bits? ¿Es –127? ¿Cuál es la representación en complemento a dos de –127? Para responder a esto, todo lo que necesitamos hacer es formar el complemento a dos de 0 1 1 1 1 1 1 1.

Paso 1.	1 0 0 0 0 0 0 0
Paso 2.	+0 0 0 0 0 0 0 1
	1 0 0 0 0 0 0 1

Todavía existe un número mayor en valor absoluto que éste, es decir 1 0 0 0 0 0 0 0 el cual es la representación en complemento a dos de –128. Así los códigos en complemento a dos de 8 bits comprenden los números entre –128 y +127.

Dado un código en complemento a dos de 8 bits, ¿cómo puede uno determinar su equivalente decimal? Aquí están algunos ejemplos más.

Ejemplo 5

¿Qué número decimal está representado mediante los siguientes números en complemento a dos de 8 bits?

a. 00001100
b. 01100001
c. 10001111
d. 11100001

A. Puesto que el código empieza con cero, representa un número positivo y nosotros interpretamos su código en complemento a dos como si fuera código binario. Así, la respuesta es 12 (base 10).

B. De nuevo tenemos un número positivo de forma que interpretamos el código binario en la forma usual y obtenemos 97 (base 10).

C. Aquí tenemos un número negativo. Para determinar qué número negativo, forme su complemento a dos y decodifíquelo:

El complemento a dos de 10001111 es 01110001

01110001 es la representación en complemento a dos de −113 (base 10)

Así, 10001111 es la representación en complemento a dos de −113 (base 10).

D. Nosotros tenemos otro número negativo así que seguimos el mismo procedimiento del apartado C.

Paso 1: Encontrar el complemento a dos de 11100001 que es 00011111.

Paso 2: Decodificar 00011111 como la representación en complemento a dos de 31 (base 10).

Paso 3: Así 11100001 es la representación en complemento a dos de −31 (en base 10).

¿Cómo se puede ir en el otro sentido? Esto es, dado un número entero decimal comprendido entre −128 y +127, ¿cómo encontramos su representación en complemento a dos? Las mismas técnicas básicas prevalecen como usted verá en el siguiente ejemplo.

Ejemplo 6

Dar la representación en complemento a dos de los siguientes números decimales.

a. 100
b. −13

A. Puesto que 100 es positivo, todo lo que tenemos que hacer es encontrar su código binario. Esto se ve fácilmente que es 01100100.

B. Esta vez tenemos un número negativo de forma que podemos aplicar aquí el viejo truco "complementar y codificar-el-positivo": en particular encontramos la representación en complemento a dos para +13 y tomamos entonces su complemento a dos.

La representación en complemento a dos para +13 = a la representación binaria para +13 = 00001101

La representación en complemento a dos de 00001101 = 11110011

Así, la representación en complemento a dos de −13 = 11110011.

El ejemplo 7 y la discusión que le acompaña son para aquellos de nuestros lectores que desean conocer un poco la teoría que está detrás de la representación en complemento a dos. Quisiéramos señalar que el entender el ejemplo 7 *no* es necesario para ser capaz de utilizar toda la considerable potencia del Z-80 para el direccionamiento indexado y direccionamiento relativo. Sin embargo, en nuestra discusión de la suma y resta en complemento a dos de los párrafos siguientes, haremos referencia a la expresión (2**n)−x, sólo para indicar que algunas de las reglas (aparentemente) arbitrarias que hemos dado hasta aquí tienen alguna justificación matemática.

En este punto, no le hemos dado a usted ninguna razón de porque tiene sentido el representar números positivos con el código binario estándar y los números negativos con el "loco" código que es el resultado de una operación de dos pasos en el código para el positivo opuesto del número. La operación de dos pasos, cambiando ceros a unos y añadiéndole entonces uno, no aparece en el cielo azul. Lo que estamos haciendo con este proceso es encontrar la representación binaria de (2**n)−x, en donde x es el número positivo con cuya representación binaria empezó usted (n es el número de bits en la representación binaria). Comprobemos esta definición con un ejemplo:

Ejemplo 7

Nosotros conocemos que 100 (base 10) tiene una representación en complemento a dos de 01100100 del ejemplo 6. Vamos ahora a encontrar el complemento a dos de 100 (base 10) utilizando la expresión (2**n)−x.

(2**n)−x=(2**8)−100=156 (base 10)

La representación binaria de 156 es 10011100.

Encontrar −100 utilizando el método de dos pasos nos lleva al mismo número binario. Usted puede comprobar esto por sí mismo.

NOTA: Si usted piensa de nuevo los resultados cuando hemos sumado un número a su complemento a dos, usted recordará que siempre hemos obtenido un 1 seguido de n ceros, en donde n era el número de bits en la representación. Desde luego, 1 seguido por n ceros es la representación binaria de 2**n. Así que todo lo que estuvimos haciendo fue sumar x y (2**n)–x para obtener 2**n.

SUMA Y RESTA EN COMPLEMENTO A DOS

Primero estudiaremos la suma. Una vez que usted pueda sumar dos números cualquiera en complemento a dos, ya habrá terminado. ¿Por qué? Porque cualquier problema de resta (x–y) puede ser reducido a un problema de suma (x+(–y)). Encontrar el complemento a dos de y, y sumarlo a x, y usted habrá realizado la resta.

La suma de dos números en complemento a dos se realiza exactamente como si los números estuvieran en representación binaria. Esta es la mayor ventaja de la notación en complemento a dos.

Ejemplo 8

```
a.   0 0 0 0 0 1 1 1   (+7)
    +0 0 0 0 0 0 1 0   (+2)
    ----------------
     0 0 0 0 1 0 0 1   (+9)
b.   1 1 1 1 1 1 0 0   (−4)
    +0 0 0 0 0 0 1 1   (+3)
    ----------------
     1 1 1 1 1 1 1 1   (−1)
c.   1 1 1 1 1 0 0 1   (−7)
    +1 1 1 1 0 0 1 1   (−13)
    ----------------
     1 1 1 0 1 1 0 0   (−20)
d.   0 1 1 0 0 0 0 0   (+96)
    +0 1 0 1 0 0 0 0   (+82)
    ----------------
     1 0 1 1 0 0 0 0   ¡CUIDADO! Dos números positivos
                              ¿dan como suma un número negativo?
e.   1 0 1 1 1 0 0 1   (−71)
     1 0 1 1 1 0 0 0   (−72)
    ----------------
     0 1 1 1 0 0 0 1   ¡CUIDADO! Dos números negativos
                              ¿dan como suma un número positivo?
```

En los dos últimos problemas de suma (d y e), tuvimos dificultades. Lo que sucedió se llama *sobrepasamiento.* Como usted recuerda los números de 8 bits en complemento a dos ocupan una gama comprendida entre –128 y +127. Cuando sumamos 96 y 82 en d, y –71 y –72 en e, nuestras sumas son 178 y –143, respectivamente. Estos son números que están fuera del límite de –128 hasta +127. Este fenómeno llamado *sobrepasamiento* en la ciencia de los computadores se produce siempre que los números están representados en códigos de una longitud fija de bits. La forma usual de tratar el sobrepasamiento es detectar cuando se produce y bifurcar a un grupo de instrucciones que impriman un mensaje de error. Así la cuestión es: ¿cómo detectar el sobrepasamiento? Para algunos códigos éste no es un problema trivial. Afortunadamente, una de las capacidades de la representación en complemento a dos es la facilidad de la detección del sobrepasamiento. Si dos números positivos dan como resultado uno negativo, o si dos números positivos suman uno negativo entonces es que existe sobrepasamiento. Comprobar esta condición se realiza fácilmente comprobando el primer bit (el bit más significativo) de cada sumando y de la suma. El Z-80 también coloca un bit (el bit P/V) en su REGISTRO DE INDICADORES, si su suma en complemento a dos produce un sobrepasamiento. El REGISTRO DE INDICADORES se discutirá en detalle en un capítulo posterior.

Tal como indicamos anteriormente, la resta se realiza complementando y sumando la cantidad que se quiere restar.

Esto concluye nuestra discusión de la representación en complemento a dos.

MODOS DE DIRECCIONAMIENTO DEL Z-80

La próxima sección cubre las grandes posibilidades de direccionamiento del Z-80. Sus diez modos de direccionamiento contribuyen en gran manera a la superioridad del Z-80 sobre el microprocesador 8080 de Intel, en términos de la riqueza de su conjunto de instrucciones. Para cada modo de direccionamiento, daremos una discusión casi exhaustiva la cual incluirá definiciones y ejemplos. Para reiterar lo que dijimos en el capítulo 6, lea lo que sigue colocando un énfasis en lo que hace el modo de direccionamiento, como se compara con otros modos de direccionamiento, y ponga mucha atención a la notación utilizada en los mnemónicos para cada modo de direccionamiento.

El esfuerzo que usted ponga en esta sección le preparará para leer y entender las tablas que son esenciales para el trabajo posterior en este libro. Al final de este capítulo le proporcionaremos varios ejercicios para ayudarle a solidificar su comprensión de estos importantes conceptos.

DIRECCIONAMIENTO POR REGISTRO

El direccionamiento por registro cuando el código de operación de una instrucción contiene información que especifica cual(es) son los registros de la CPU es/están implicados en la ejecución de la instrucción. Los códigos de operación que contienen los códigos del registro de tres bits señalados en el capítulo 6 son ejemplos de este tipo de direccionamiento. Considera la instrucción:

LD A,B

cuyo código de operación es: $0\ 1\ \underset{\text{A}}{\underline{1\ 1\ 1}}\ \underset{\text{B}}{\underline{0\ 0\ 0}}$ o 78 hex.

El direccionamiento por registro se implementa dos veces en esta instrucción, primero para el registro A y en segundo lugar para el registro B.

DIRECCIONAMIENTO INMEDIATO

El modo de *direccionamiento inmediato* es utilizado con instrucciones multibyte que contiene el byte de datos de 8 bits con el cual se debe operar. La siguiente instrucción de carga utiliza el direccionamiento inmediato:

LD C,03H

cuyo código asociado es: 0E 03. La ejecución de esta instrucción concluye con un código hex 03 colocado en el registro C de la CPU. (¿Utiliza esta instrucción otro tipo de modo de direccionamiento? El código de operación indica que el registro C debe de ser cargado, así que se utiliza el modo de direccionamiento por registro.)

DIRECCIONAMIENTO INMEDIATO EXTENDIDO

Este modo de direccionamiento requiere que la instrucción proporcione dos bytes de dato siguiendo inmediatamente al código de operación en lugar de 1 byte que se necesita para direccionamiento inmediato. Así, este modo "extiende" el modo de direccionamiento inmediato. Claramente el código máquina para cualquier instrucción que utilice este modo de direccionamiento es por lo menos de una longitud de tres bytes con un byte para el código de operación y dos bytes para el dato. La instrucción:

LD BC,0421H

cuyo código hex asociado es 01 21 04 utiliza direccionamiento inmediato extendido. Asegúrese de anotar el orden en el cual los bytes de dato aparecen en el código máquina para esta instrucción. El byte para el registro C (21) es el byte LO y, así, viene en primer lugar. Esto es cierto para todas las cargas de pares de registros.

DIRECCIONAMIENTO INDIRECTO POR REGISTRO

Hemos visto modos para direccionar registros y modos en los cuales el dato es parte de la instrucción. El modo de *direccionamiento por registro indirecto* utiliza un par de registros para indicar en que parte de la memoria reside el dato. Esto es, el par de registros contiene la dirección del dato que necesita la instrucción. Una instrucción que utiliza el direccionamiento por registro indirecto es:

```
LD A, (HL)
```

cuyo código hex asociado es: 7E. Para mostrar que el contenido del par de registros HL se utiliza como un indicador de una posición de memoria, HL está incluido entre paréntesis. Esta notación es estándar para el direccionamiento por registro indirecto.

Para algunas instrucciones, el direccionamiento indirecto se utiliza para especificar dos bytes con los cuales operará la instrucción. En tales casos, el contenido del par de registros especifica el byte LO y el contenido más uno señala al byte HI. Por ejemplo la instrucción

```
POP BC
```

cuyo código hex asociado es: C1 carga (SP) en C y (SP + 1) en B.

DIRECCIONAMIENTO EXTENDIDO

Una instrucción que utiliza *direccionamiento extendido* contiene, como sus dos últimos bytes, una dirección de 16 bits. Esta dirección se puede utilizar como indicador de una posición de memoria para el dato requerido o también puede ser la dirección a la cual debe saltar el programa. Un ejemplo del último uso es,

```
LD (1203H),A
```

cuyo asociado código hex es: 32 03 12. Esta instrucción hace que la posición de memoria 1203 sea cargada con el contenido del acumulador. Obsérvese que, en consonancia con el direccionamiento por registro indirecto, la dirección está incluida entre paréntesis. La notación generalizada para esto es (nn), en donde n es un byte de 8 bits.

Una instrucción en la cual nn representa una dirección a la cual debe saltar el programa es la instrucción JP nn, por ejemplo:

```
JP 1203H
```

cuyo asociado código hex es: C3 03 12. Obsérvese que en esta instrucción no estamos transfiriendo datos, si no por el contrario control del programa a la instrucción situada en la dirección 1203. Así, esta vez nn no está incluido entre paréntesis.

DIRECCIONAMIENTO MODIFICADO PAGINA CERO

Hay ocho instrucciones del Z-80 que utilizan direccionamiento modificado página cero. Estas instrucciones, llamadas instrucciones de *restart,* provocan que el control del programa sea transferido a una sección del programa llamado una *subrutina.* Discutiremos este tipo de transferencia del control del programa más tarde. Todas las instrucciones de restart son de una longitud de un solo byte. El código de operación especifica cualquiera de las ocho posibles direcciones –0000, 0008, 0010, 0018, 0020, 0028, 0030 o 0038– una para cada instrucción de restart. Como que su dirección alta es siempre 00, el modo de direccionamiento es llamado *modificado página cero.* El propósito principal de la instrucción de restart es el acceder a subrutinas que son utilizadas muy a menudo. La ventaja de las instrucciones de restart son las de que ahorran tiempo y espacio y que pueden ser accedidas en el chip del microprocesador durante una interrupción. Las instrucciones de "llamada" a subrutina comparables utilizan tres bytes en lugar de uno, que es el que necesita un restart. Aquí está un ejemplo de una instrucción de restart:

```
RST 10H
```

cuyo código hex asociado es: D7. Esta instrucción transfiere el control del programa a la subrutina situada en 0010 (decimal 16).

DIRECCIONAMIENTO IMPLICITO

Ciertas instrucciones del Z-80 se aplican automáticamente a un registro determinado. Esta clase de instrucciones utilizan *direccionamiento implícito.* El grupo de instrucciones aritméticas y lógicas de 8 bits son ejemplos de instrucciones de direccionamiento implícito porque todas ellas realizan operaciones con el contenido del acumulador. La instrucción

```
ADD A,B
```

cuyo código hex asociado es: 80, suma el contenido del registro B al acumulador y carga el acumulador con la suma.

DIRECCIONAMIENTO DE BIT

El conjunto de instrucciones del Z-80 contiene muchas instrucciones que direccionan bits individuales dentro de un byte guardado en la memoria o en los registros. Estas instrucciones de manipulación de bits utilizan una combinación de modos de direccionamiento registro, registro indirecto, o direccionamiento indexado que especifica la posición de memoria o el registro de la CPU del byte implicado; un código de tres bits dentro del código de operación de la instrucción especifica el bit –bit 0, 1, 2, 3, 4, 5, 6 o 7– en donde los bits están numerados desde la derecha a la izquierda (menor peso a mayor peso) dentro de un byte:

Byte	MSB							LSB
Número de bit	7	6	5	4	3	2	1	0

Un ejemplo de instrucción de direccionamiento de bit es

SET 3,B

cuyo código hex asociado es CB DB. Esta instrucción "coloca" (SET) al valor lógico 1 el bit número 3 decimal del registro B. Muchas de las instrucciones del nuevo Z-80 no implementadas en el microprocesador Intel 8080 son las instrucciones BIT, SET y RESET que utilizan todas ellas el modo de direccionamiento por bit.

DIRECCIONAMIENTO INDEXADO

El Z-80 tiene dos registros de 16 bits llamados *registros índice* que tienen un propósito especial. Son llamados los registros IX e IY. Su principal utilización es para el modo de direccionamiento indexado. El direccionamiento indexado es muy similar al direccionamiento por registro indirecto puesto que el contenido de un registro de 16 bits señala a la posición de memoria del dato deseado. Una diferencia importante es que para el direccionamiento indexado, se debe especificar un byte de desplazamiento en el primer byte de la instrucción después del código de operación. Este byte es un número de 8 bits en complemento a dos, el cual indica cuantos bytes más arriba o más abajo en la memoria desde la dirección contenida en el registro índice está la posición del byte con el que se debe operar. Por ejemplo,

LD A,(IX+02H)

cuyo código hex asociado es: DD 7E 02, carga el acumulador con el contenido de la posición de memoria situada dos bytes más lejos que la posición señalada por IX. La instrucción

LD (IY+FFH),A

cuyo código asociado es FD 77 FF carga el contenido del acumulador en la posición de memoria menor de uno que la dirección en el registro IY, puesto que FF

es la representación en complemento a dos del número decimal −1. La tabla 7-1 muestra las instrucciones LD precedentes.

Tabla 7-1. Instrucciones LD

(IX+d)	Significado del desplazamiento con respecto a la posición de memoria (en decimal)
(IX+7FH)	127 bytes mayor que IX
(IX+0FH)	15 bytes mayor que IX
(IX+09H)	9 bytes mayor que IX
(IX+01H)	1 byte mayor que IX
(IX+00H)	(IX)
(IX+FFH)	1 byte menor que IX
(IX+FEH)	2 bytes menor que IX
(IX+FDH)	3 bytes menor que IX
(IX+FCH)	4 bytes menor que IX
(IX+F0H)	16 bytes menor que IX
(IX+D0H)	32 bytes menor que IX
(IX+C0H)	48 bytes menor que IX
(IX+80H)	128 bytes menor que IX

La notación para indicar direccionamiento indexado es (IX + d) o (IY + d) en donde d representa el byte de desplazamiento en complemento a dos. El paréntesis indica que IX + d y IY + d son indicadores que señalan a una posición de memoria. La figura 7-1 resume el significado de d en la instrucción que utiliza direccionamiento indexado tal como LD A,(IX + d).

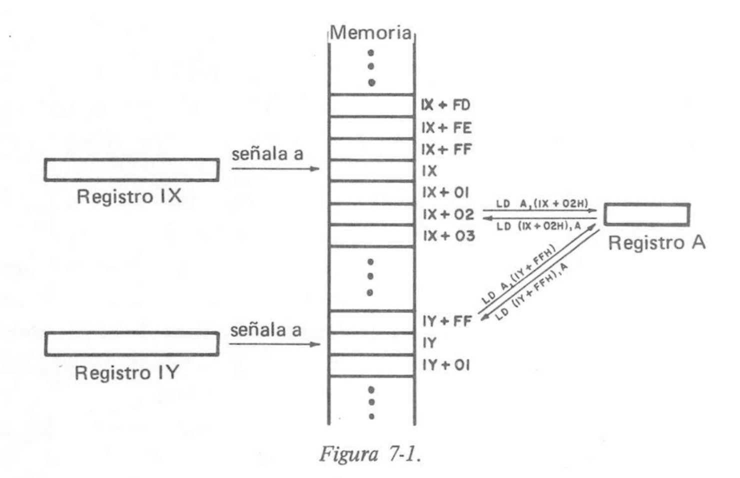


Figura 7-1.

El direccionamiento indexado es una potente herramienta para acceder a tablas de datos en la memoria. Típicamente, el registro IX o IY es cargado con la direc-

ción de la primera entrada de la tabla y entonces todas las otras entradas a la tabla son referidas a su posición relativa a la primera entrada. Esto es, el byte de desplazamiento es cambiado en forma apropiada de acuerdo con la entrada a la tabla accedida. Esto ilustra el hecho importante de que la ejecución de una instrucción que utiliza direccionamiento indexado no cambia el contenido del registro índice. Al final de este capítulo aparecen varios programas que utilizan direccionamiento indexado para acceder a una tabla.

DIRECCIONAMIENTO RELATIVO

El *direccionamiento relativo* es un modo de direccionamiento muy especializado que se aplica a las instrucciones de salto llamados *saltos relativos* (JR). Tal como sucede con el direccionamiento indexado, el primer byte después del código de operación es un número en complementos a dos que representa un desplazamiento desde alguna dirección. Considere la instrucción

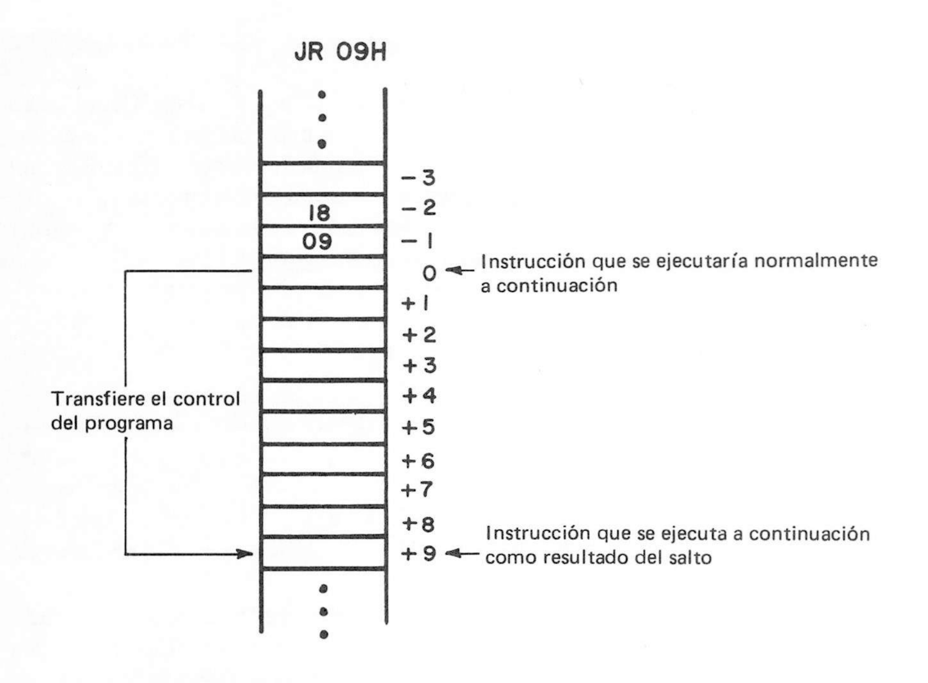
```
JR 09H
```

cuyo código hex asociado es: 18 09. El 09 es el desplazamiento desde la dirección de la próxima instrucción a la *instrucción que se va a ejecutar a continuación.* Esto es, este es un salto incondicional relativo a una instrucción situada nueve bytes más adelante en el programa a partir de la instrucción que normalmente sería ejecutada a continuación. La instrucción

```
JR FCH
```

cuyo código hex asociado es 18 FC provoca que el control del programa sea transferido cuatro bytes hacia atrás desde la próxima instrucción puesto que FC es la representación de ocho bits en complemento a dos para –4. En la figura 7-2 aparece una ilustración de estas dos instrucciones.

El modo de direccionamiento relativo del Z-80 permite una capacidad de programación muy importante y es la de poder escribir código *reubicable.* Se dice que un programa o bloque de instrucciones es relocatable o reubicable si es independiente de donde reside físicamente en la memoria. Para probar si un programa es reubicable, se mueve el programa sin cambiarlo a una nueva posición de la memoria. Si el programa se ejecuta correctamente entonces el programa es reubicable. Está claro que cualquier programa, que utilice direccionamiento extendido, no es reubicable. Un salto normal especifica una dirección absoluta, de forma que moviendo el programa a una nueva posición requiere que se cambie esta dirección absoluta antes de una ejecución correcta. El proceso de cambiar todas las direcciones absolutas de acuerdo con el cambio de la posición de un programa es llamado *reubicar* el programa. Otra ventaja de los saltos relativos es que solamente necesita dos bytes de


JR 09H
18
09
-3
-2
-1
0
+1
+2
+3
+4
+5
+6
+7
+8
+9
Instrucción que se ejecutaría normalmente a continuación
Transfiere el control del programa
Instrucción que se ejecuta a continuación como resultado del salto

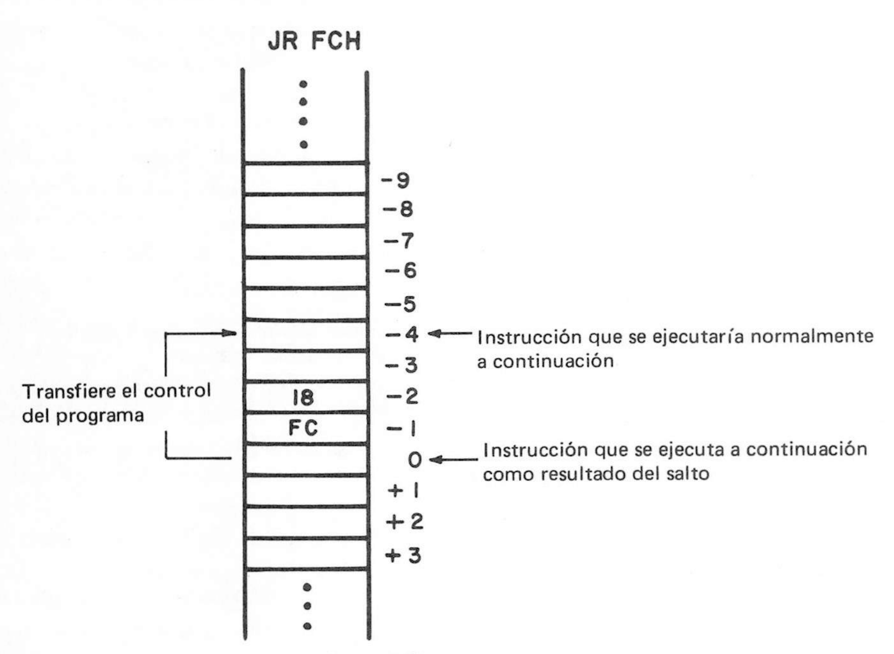

JR FCH
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
+1
+2
+3
18
FC
Instrucción que se ejecutaría normalmente a continuación
Transfiere el control del programa
Instrucción que se ejecuta a continuación como resultado del salto

Figura 7-2.

memoria mientras que los saltos absolutos necesitan tres bytes, pero su gama está limitada a +127 y –127 bytes.

Esto concluye la lista de los modos de direccionamiento del Z-80. Como usted indudablemente ha apreciado, se han introducido un buen número de nuevas instrucciones. Pensamos que cada modo de direccionamiento debe ser ilustrado con un ejemplo como mínimo, aunque ello signifique introducirle a una nueva instrucción. Esté confiado porque volveremos a discutir cada una de estas nuevas instrucciones con discusiones exhaustivas en los capítulos sucesivos.

LAS TABLAS DE GRUPOS DE INSTRUCCIONES

Ahora que usted conoce todos los modos de direccionamiento del Z-80, queremos introducirle a un método extremadamente útil para mostrar las instrucciones del Z-80 con su código máquina hex asociado. Las tablas de grupos de instrucciones aparecieron por primera vez en el Manual Técnico de la CPU del Zilog Z-80. Vamos en primer lugar a examinar la tabla del grupo de instrucciones de carga de 8 bits, tabla 7-2.

Obsérvese que las filas por debajo del lado de la izquierda así como las columnas en la parte alta están señaladas con modos de direccionamiento. Hay dos modos de direccionamiento utilizados por cada instrucción de carga de 8 bits: uno para el destino (filas) y uno para el origen (columnas). Suponga que usted desea mover el contenido del registro C al registro D. Entonces D es el registro de destino de forma que usted localiza la fila (horizontal) denominada D en la tabla.

Busque en las columnas hasta que encuentre la columna del registro de origen C y usted encontrará el código máquina hex que corresponde a la instrucción LD D,C el cual es 51. En cada celda de la tabla aparece un código hex para el cual existe una instrucción del Z-80. Así, esta tabla le dice a usted cuáles son las instrucciones que están implementadas, así como su código hex asociado. Vamos a mirar algunos ejemplos.

LD A, (IX+d) tiene el código hex DD 7E d, en donde el tercer byte d es el desplazamiento, en esta aplicación de direccionamiento indexado.

LD (nn),A tiene el código hex 32 nn, en donde la primera n es el byte LO y la segunda n es el byte HI de la dirección que se va a cargar en el contenido de A

LD (IY+d),n tiene el código hex FD 36 d n, en donde n es el desplazamiento y n es el byte que se cargará en la posición de memoria desplazada de d bytes de la posición IY.

LD (HL),(BC) no está implementada en el Z-80.

Obsérvese que ciertos modos de direccionamiento no aparecen como etiquetas para las filas o columnas. Si un modo de direccionamiento no aparece en la tabla

Tabla 7-2. El grupo de carga de 8 bits

DESTINO	ORIGEN	IMPLICITO		REGISTRO							REG. INDIRECTO			INDEXADO		DIREC. EXT	INME
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
REGISTRO	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG. INDIR.	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDEXADO	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
DIREC EXT	(nn)			32 n n													
IMPLICITO	I			ED 47													
	R			ED 5F													

Cortesía Zilog Inc.

para un grupo de instrucciones, este grupo no utiliza este modo de direccionamiento. Así, podemos ver allí que existen cuatro modos de direccionamiento que no están implementados en el grupo de carga de 8 bits: bit, relativo, inmediato extendido, y modificado página cero.

EL GRUPO DE CARGA DE 16 BITS

La tabla del grupo de instrucciones de carga de 16 bits aparece en la tabla 7-3. Este está mucho menos poblado de instrucciones que la tabla del grupo de carga de 8 bits en la tabla 7-2. Muchas de las instrucciones de carga de 16 bits incluyen o bien direccionamiento inmediato extendido o direccionamiento extendido con muy pocas transferencias de 16 bits entre pares de registros. Existe solamente un par de registros para el cual es posible el direccionamiento por registro indirecto. Este registro, el *indicador de stack* SP, tiene una función muy especial que discutiremos ahora con detalle.

Tabla 7-3. El grupo de carga de 16 bits "LD", "PUSH" y "POP"

		ORIGEN									
		REGISTRO							INM. EXT.	DIR EXT.	REG. IND.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
DESTINO REGISTRO	AF										F1
	BC								01 n n	ED 4B n n	C1
	DE								11 n n	ED 5B n n	D1
	HL								21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
DIR EXT.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
INSTRUCCIONES PUSH → REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5			

NOTA: Las instrucciones Push y Pop ajustan el SP después de cada ejecución

↑ INSTRUCCIONES POP

Cortesía de Zilog, Inc.

PUSH y POP el stack

El stack (*pila*): En la ciencia de los computadores, la palabra *stack* se refiere a una estructura de datos, o forma de guardar los datos, que tiene la siguiente analogía con "la vida de cada día":

> La escena es una cafetería. Los platos limpios para ser utilizados por los clientes se colocan en una pila o stack en el mostrador. La forma más conveniente de utilizar un plato es coger el que está en lo alto de la pila. Así a medida que se va sirviendo a nuevos clientes, se retiran desde lo alto del stack. Cuando los platos utilizados se han lavado y secado, son empujados o apilados

en lo alto del stack. La relación crítica a observar al utilizar y restablecer los items del stack es:

ULTIMO DENTRO, PRIMERO FUERA

Esta regla (LIFO) (de "last in first out") es lo que caracteriza los stacks como estructuras de datos en la ciencia de los computadores. Vamos a ilustrar este nuevo concepto con un ejemplo utilizando bytes de la memoria del computador en lugar de platos. La figura 7-3 muestra una sección de la memoria en donde cada posición

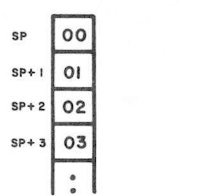


Figura 7-3.

está señalada con su dirección. Nótese que en esta discusión de las operaciones del stack, las direcciones de memoria se *incrementan* a medida de que usted mira hacia *abajo* en la página. Esta es una diferencia del tratamiento normal de los diagramas de la memoria. Hacemos esto debido a que el indicador de stack, registro SP, siempre señala al byte en el stack con la dirección menor. Este conjunto de posiciones de memoria se puede ver como una pila con una posición SP (el *indicador de stack*) que representa la dirección del byte superior. El contenido de la parte alta del stack, (SP), se muestra como 00.

Se pueden realizar dos operaciones en esta pila de bytes:

1. EXTRAER (POP) bytes de lo alto.
2. INTRODUCIR (PUSH) nuevos bytes dentro del stack.

Estas dos operaciones producen un nuevo byte situado en lo alto del stack. El microprocesador Z-80 tiene dos instrucciones, POP y PUSH, las cuales realizan los puntos 1 y 2 anteriores para dos bytes a la vez. Ambas instrucciones necesitan que se especifique un par de registros como el origen (para PUSH) o el destino

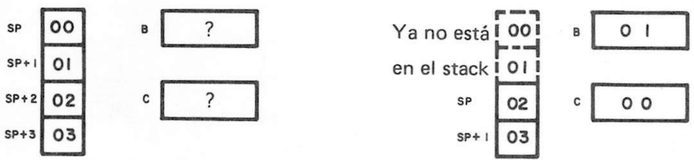


Figura 7-4.

(para POP) de los bytes de datos que se están transfiriendo. Los ejemplos de la figura 7-4 deben aclarar esto. El hecho más importante a recordar es que el *byte de lo alto del stack tiene la dirección menor.*

Ejemplo 1

La instrucción POP BC se ilustra en la figura 7-4.
La ejecución de la instrucción POP BC tiene los siguientes efectos:

1. El byte de lo alto del stack (SP) es cargado en el registro C

   C ←——— (SP)

2. El segundo byte del stack (SP + 1) es cargado en el registro B

   B ←——— (SP + 1)

3. El indicador de stack (registro SP) se actualiza para señalar a la nueva parte alta del stack, eliminando así a los dos bytes 00 y 01 del stack. Este cambio del SP se alcanza sumando 2 al indicador de stack original para llegar al nuevo valor.

   SP ←——— (SP) + 2

Así, las operaciones POP provocan que el indicador de stack se *incremente.* Obsérvese que aunque los bytes 00 y 01 permanecen en donde estaban antes de la ejecución de POP BC, la *posición* del stack en la memoria ha cambiado de tal forma que han quedado excluidos. Esto representa una diferencia sutil entre POP bytes y POP platos en una cafetería puesto que los platos se quitan físicamente de la pila o stack.

Ejemplo 2

La instrucción PUSH HL es demostrada en la figura 7-5:

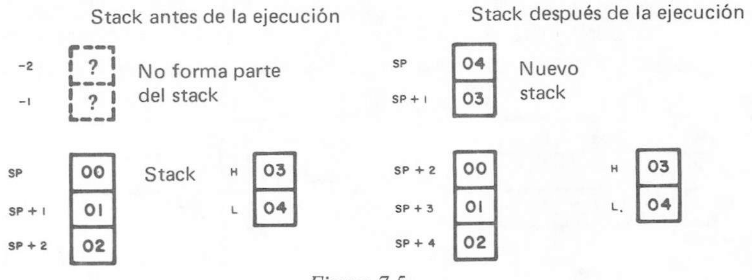


Figura 7-5.

Como usted puede ver, PUSH HL tiene el efecto opuesto de la instrucción POP:

1. El byte en el registro H es cargado en la posición de memoria situada una más desde la parte superior del stack, SP-1.

   (SP – 1)←——H

2. El byte del registro L es cargado en la posición de memoria situada dos lugares más arriba desde la parte superior stack, SP-2.

   (SP – 2)←——L

3. El indicador de stack se actualiza para señalar a la nueva parte superior del stack.

   SP←——SP – 2

   (Se le restan dos al indicador de stack para obtener el nuevo indicador de stack).

Así, las operaciones PUSH provocan que el indicador de stack *¡disminuya!* Ambos ejemplos ilustran algunos hechos que son muy importantes a recordar:

A. El stack crece desde las direcciones altas a las bajas en la memoria. Esto es, "el stack crece *hacia abajo* en la memoria". POP *incrementa* el SP y PUSH *decrementa* el SP.
B. Siempre se introducen o extraen dos bytes. Todas las operaciones de PUSH y POP tienen lugar entre el stack y pares de registros o registros índice: AF, BC, DE, HL, IX o IY. Los bytes salen desde lo alto del stack con el byte LO en primer lugar y a continuación el byte HI. En la operación PUSH el byte HI en primer lugar y a continuación el byte LO.
C. Las instrucciones PUSH y POP difieren de una carga normal de 16 bits porque la transferencia de datos está acompañada de una actualización del registro indicador de stack.
D. Las instrucciones PUSH y POP utilizan direccionamiento indirecto por registro porque la posición de memoria del dato viene señalada por el contenido del registro SP de 16 bits.

El stack y sus operaciones asociadas, PUSH y POP, son utilizados en su mayor parte en conjunción con una transferencia del control del programa denominado *llamada a subrutina.* El capítulo que trata de saltos, llamadas y retornos cubre este sujeto. Aplazaremos las discusiones del funcionamiento del stack hasta este capítulo. Nuestro propósito al introducir aquí las operaciones del stack es el de que usted pueda tener una comprensión completa del grupo de instrucciones de carga de 16 bits del Z-80.

TRANSFERENCIA DE BLOQUES E INTERCAMBIOS

Antes de acabar este capítulo, presentaremos dos tablas más de reducido tamaño: las transferencias de bloques y los intercambios en las tablas 7-4 y 7-5 respectivamente.

Tabla 7-4. Grupo de transferencia de bloques

			ORIGEN: REG. INDIR. (HL)	
DESTINO	REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)←(HL) Inc HL & DE, Dec BC
			ED B0	'LDIR,' – Load (DE)←(HL) Inc HL & DE, Dec BC, Repetir hasta que BC = 0
			ED A8	'LDD' – Load (DE)←(HL) Dec HL & DE, Dec BC
			ED B8	'LDDR' – Load (DE)←(HL) Dec HL & DE, Dec BC, Repetir hasta que BC = 0

Reg HL señala al origen
Reg DE señala al destino
Reg BC es el contador de byte

Cortesía de Zilog, Inc.

Para una discusión de las transferencias de bloques le enviamos al capítulo 6.

Las instrucciones de intercambio efectúan un "canje" de bytes de datos entre registros de 16 bits o pares de registros. Por ejemplo,

EX DE,HL

cuyo código asociado es EB, *intercambia* el contenido de DE con el contenido de HL:

Antes de la ejecución de EX DE,HL		Después de la ejecución de EX DE,HL	
D	00	D	02
E	01	E	03
H	02	H	00
L	03	L	01

La instrucción, EX (SP),HL intercambia el contenido de H y L con los dos bytes de lo alto del stack; existen instrucciones similares para los registros índices. Los

Tabla 7-5. Intercambios "EX" y "EXX"

		DIRECCIONAMIENTO IMPLICITO				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLICITO	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

Cortesía de Zilog, Inc.

intercambios EXX y EX AF,AF' son las únicas instrucciones del Z-80 que incluyen el segundo conjunto de registros de uso general B', C', D', E', H', L' y F'. Así, usted puede ver que estos registros alternos se pueden utilizar como almacenamiento temporal para los registros principales y no pueden ser accedidos con flexibilidad.

INTRODUCCION A LOS EXPERIMENTOS Y EJERCICIOS

Hemos incluido experimentos y ejercicios al final de esta unidad para ayudarle a solidificar su comprensión de la representación binaria en complemento a dos, los modos de direccionamiento del Z-80, operaciones del stack, y la utilización de las tablas de instrucciones del Z-80. Le recomendamos que efectúe algunos de los ejercicios antes de realizar los experimentos. Así, hemos colocado los ejercicios de repaso antes de los experimentos para animarle a hacerlo.

Los experimentos que usted realizará se pueden resumir de la siguiente forma.

Experimento N.°	Comentarios
1	Demuestra la manipulación de tablas mediante direccionamiento indexado.
2	Demuestra otros métodos para realizar manipulación de tablas. Uno de los ejemplos es un programa que se automodifica.

3 Demuestra las operaciones del stack, PUSH y POP y las instrucciones de intercambio.

REPASO

1. Encontrar el complemento a dos de 8 bits de los siguientes números binarios de 8 bits:

   a. 0 0 0 0 0 0 0 1
   b. 1 1 0 1 1 0 1 0
   c. 0 1 0 1 0 1 0 1
   d. 1 1 1 0 1 1 1 0
   e. 0 0 0 0 1 1 1 0
   f. 1 0 0 0 0 0 0 0
   g. 1 1 1 1 1 1 1 1

2. a. ¿Cuál es el mayor número decimal entero que tiene una representación en forma de 8 bits en complemento a dos?
   b. ¿Cuál es el mayor número (en valor absoluto), decimal y negativo que tiene una representación como un número de ocho bits en complemento a dos?
   c. Conteste a y b para números en complemento a dos de 16 bits.
3. Encontrar el número decimal representado por los siguientes números de 8 bits en complemento a dos.

   a. 0 1 1 1 1 0 0 0
   b. 1 0 1 0 0 0 1 1
   c. 0 0 0 0 0 0 1 1
   d. 1 1 1 1 1 1 1 1
   e. 1 1 1 1 0 0 1 1
   f. 0 1 0 1 0 1 0 0
   g. 1 1 0 1 1 0 0 1

4. Encontrar la representación de 8 bits en complemento a dos de los siguientes números decimales.

   a. 1
   b. 16
   c. −16
   d. −128
   e. 128
   f. 121
   g. −90

5. La siguiente es una lista de las instrucciones de salto relativo con su código máquina hex asociado. Utilice esta información para convertir todas las instrucciones de saltos (JP) "absolutos", a instrucciones de salto relativo (JR) en los siguientes programas.

Instrucciones de salto relativo	Operación en código hex
JR	18
JR NZ	20
JR Z	28
JR PE	no implementado

   a. Programa N.° 9 en el Experimento N.° 3 del capítulo 6.
   b. Programa N.° 10 en el Experimento N.° 3 del capítulo 6.
   c. Programa N.° 12 en el Experimento N.° 5 del capítulo 6.
6. Para cada una de las siguientes instrucciones, dé los modos de direccionamiento utilizados y el código hex asociado utilizando las tablas del Z-80.

   a. LD A,B
   b. JR FBH (el Ejercicio 5 lo da el código hex)
   c. LD A,(IX+06H)
   d. LD (IX+06H),A
   e. LD (1234H),A
   f. LD (IX+09H),33

g. LD SP,HL
h. LD BC,0109H
i. LD (1030H),BC
j. LD IX,(1000H)
k. PUSH BC
l. POP IX

7. Indicar cuales de las siguientes instrucciones están implementadas en el Z-80. Si es así, dar el código hex asociado.

a. LD AF,BC
b. LD B,(BC)
c. LD (BC),B
d. LD IX,IY
e. LD HL,BC
f. LD (1234H),56H
g. LD (1234H),B
h. LD (DE), 45H
i. PUSH 1234H
j. POP SP

Respuestas

1. a. 1 1 1 1 1 1 1 1
   b. 0 0 1 0 0 1 1 0
   c. 1 0 1 0 1 0 1 1
   d. 0 0 0 1 0 0 1 0
   e. 1 1 1 1 0 0 1 0
   f. no existe ninguno
   g. 0 0 0 0 0 0 0 1
2. a. 0 1 1 1 1 1 1 1 = +127(base 10)
   b. 1 0 0 0 0 0 0 0 = –128(base 10)
   c. 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = (2**15–1) (base 10) = mayor
   d. 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = (–2**15) (base 10) = menor
3. a. 120
   b. –93
   c. 3
   d. –1
   e. –13
   f. 84
   g. –39
4. a. 0 0 0 0 0 0 0 1
   b. 0 0 0 1 0 0 0 0
   c. 1 1 1 1 0 0 0 0
   d. 1 0 0 0 0 0 0 0
   e. no existe ninguno
   f. 0 1 1 1 1 0 0 1
   g. 1 0 1 0 0 1 1 0
5. a.

Posición de memoria	Código objeto		Código fuente
0120	0E 00		LD C,00H
0122	0D	LOOP:	DEC C
0123	20 FD		JR NZ,LOOP
0125	FF		RST 38H

Para determinar la dirección relativa que se debe utilizar como <B2> en la instrucción JR NZ, utilice la siguiente ecuación:

dirección relativa = complemento a dos de 8 bits de (la dirección absoluta de la instrucción después de la instrucción de salto relativo menos la dirección absoluta del destino del salto)
= complemento a dos de 8 bits de (0125-0122)*
= complemento a dos de 8 bits del byte 03
= 1 1 1 1 1 1 0 1
= FD

* Obsérvese que la diferencia entre estas dos direcciones hex de 16 bits deben tener una representación en complemento a dos de ocho bits para el salto relativo que va a ser definido entre las dos direcciones.

Este programa se acortó de un byte reemplazando un salto absoluto por uno relativo.

5. b.

Posición de memoria	Código objeto	Código fuente
0130	06 00	LD B,00H
0132	0E 00	LOOP1:LD C,00H
0134	0D	LOOP2:DEC C
0135	20 FD	JR NZ,LOOP2
0137	05	DEC B
0138	20 F8	JR NZ,LOOP1
013A	FF	RST 38H

Para la instrucción JR NZ,LOOP2:
dirección relativa = complemento a dos (0137-0134)
= complemento a dos de 03
= FD
Para la instrucción JR NZ,LOOP1:
dirección relativa = complemento a dos de (013A-0132)
= complemento a dos de (08)
= F8

Este programa ha sido acortado de dos bytes reemplazando dos saltos absolutos por dos saltos relativos.

5. c.

Posición de memoria	Código objeto	Código fuente
0180	21 A0 01	LD HL,01A0H
0183	11 C0 01	LD DE,01C0H
0186	01 10 00	LD BC,0010H
0189	7E	LOOP: LD A,(HL)
018A	B7	OR A
018B	28 05	JR Z,QUIT
018D	ED A0	LDI
018F	EA 89 01	JP PE,LOOP
0192	FF	QUIT: RST 38H

De los dos saltos que hay en este programa, solamente uno JP Z,QUIT, puede ser convertido a un salto relativo. La instrucción JE PE no tiene un equivalente a salto relativo en el conjunto de instrucciones del Z-80. Así para la instrucción JP Z,QUIT:
dirección relativa = complemento a dos (dirección después del salto menos la dirección de destino)
= complemento a dos (018D-0192)
= (0192-018D)
= 05
Nótese que para saltos a direcciones mayores, el cálculo es más fácil porque se evita la operación de buscar el complemento a dos cambiando el orden de la resta de las direcciones (el complemento a dos de A-B es igual a B-A). También el complemento a dos del complemento a dos de A es A.

6. a. Direccionamiento de registro para fuente y destino–Código hex: 78
b. Direccionamiento relativo–Códifo hex 18 FB
c. Destino: direccionamiento por registro–Código hex: DD 7E 06
Fuente: direccionamiento indexado–(IX + d)
d. Destino: direccionamiento indexado–(IX + d)–Código hex: DD 77 06
Fuente: direccionamiento por registro

e. Destino: direccionamiento extendido–(nn)–Código hex 32 34 12
Fuente: direccionamiento por registro
f. Destino: direccionamiento indexado–(IX + d)–Código hex: DD 36 09 33
Fuente: direccionamiento inmediato–n
g. Destino: direccionamiento por registro–Código hex: F9
Fuente: direccionamints por registro
h. Destino: direccionamiento por registro–Código hex 01 09 01
Fuente: inmediato extendido–nn
i. Destino: direccionamiento extendido–(nn)–Código hex: ED 43 30 10
Fuente: direccionamiento por registro
j. Destino: direccionamiento por registro–Código hex DD 2A 00 10
Fuente: direccionamiento extendido–(nn)
k. Destino es (SP) y (SP-1): direccionamiento indirecto por registro–Código hex 05
El origen es el par de registros BC: direccionamiento por registro
l. El destino es IX: direccionamiento por registro–Código hex: DD E1
La fuente es (SP) y (SP + 1): direccionamiento por registro indirecto

7. Ninguna de estas instrucciones están implementadas.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar la manipulación de tablas mediante direccionamiento indexado.

Programa N.° 15

Posición de memoria	Código objeto		Código fuente	Comentarios
0100	01 03 00		LD BC,0003H	; 3 bytes por línea
0103	FD 21 20 01		LD IY,0120H	; Dirección de inicio de la tabla = 0120
0107	FD 7E 00	LOOP:	LD A,(IY)	; Cargar columna 1 a A
010A	B7		OR A	; ¿Es cero?
010B	28 0A		JR Z,END	; Si es así, final
010D	FD 86 01		ADD (IY+01H)	; Si no sumar columna 2
0110	FD 77 02		LD (IY+02H),A	; Guardar la suma en la columna 3
0113	FD 09		ADD IY,BC	; IY señala a la próxima línea
0115	18 F0		JR LOOP	; Repetir el procedimiento anterior
0117	FF	END:	RST 38H	; Devolver el control al sistema
				; operativo.

Paso 1

Cargar el programa precedente empezando en la posición 0100. Verifique que usted lo ha cargado correctamente.

Paso 2

Este programa manipula una tabla que está hecha de filas de tres bytes de largo. Para cada fila o línea de la tabla, las dos primeras columnas se suman y la suma es guardada en la columna tercera. Este proceso continúa hasta que se encuentra una línea cuyo primer byte es 00. En este momento, se devuelve el control al sistema operativo del Nanocomputador. Para simplificar el problema, supondremos por ahora que los sumandos en las columnas 1 y 2 son suficientemente pequeños para que no exista la posibilidad de sobrepasamiento en la suma. Considere el siguiente diagrama de una tabla en la memoria:

	Dirección	Col 1	Col 2	Col 3
Línea 1	0120	01	02	?
Línea 2	0123	10	04	?
Línea 3	0126	23	13	?
Línea 4	0129	06	24	?
Línea 5	012C	00		

IY es colocado inicialmente a 0120 e incrementado de 0003 para cada nueva línea en secuencia.

Paso 3

Inicializar las posiciones de memoria desde 0120 hasta 012C con los valores que aparecen en la tabla anterior.

Paso 4

Ejecutar el programa en modo paso a paso observando lo que sucede a los registros IY y A, así como a las posiciones de memoria 0122, 0125, 0128 y 012B.

Ahora usted ha apreciado ciertamente lo adecuado que resulta el direccionamiento indexado para la manipulación de dos tablas dimensionales de información. La línea en la tabla es establecida mediante el contenido del registro IY mientras que las entradas en cada línea se especifican mediante el desplazamiento desde IY.

EXPERIMENTO N.° 2

Propósito

El propósito de este experimento es el de mostrarle que pueden existir varias

formas alternativas de escribir un programa para realizar una determinada tarea. Una técnica es la automodificación, en la cual el programa modifica sus propias instrucciones a medida que se ejecuta. CUIDADO: NO ESTAMOS RECOMENDANDO QUE USTED ADOPTE ESTA TECNICA PERO DEBE TENER CONOCIMIENTO DE SU EXISTENCIA.

Programa N.° 16

Posición de memoria	Código objeto	Código fuente	Comentarios
02FD	01 07 00	LD BC,0007H	; BC = al número de columnas por fila
0300	1E 06	LD E,06H	; El registro E se utilizará como ; un contador del número de líneas ; procesadas
0302	FD 21 80 03	LD IY,0380H	; IY señala a la línea que se está ; procesando actualmente
0306	21 11 03	LD HL,0311H	; HL señala a la posición del ; byte de desplazamiento en la ; instrucción ADD(IY + d)
0309	36 00	ROW: LD (HL),00H	; Inicializa el desplazamiento
030B	3E 00	LD A,00H	; Inicializa el registro A
030D	16 06	LD D,06H	; El registro D cuenta el número ; de columnas sumadas
030F	FD 86 d	COL: ADD A,(IY+d)	; Aquí utilizamos d porque el ; desplazamiento cambia a medida ; que se ejecuta el programa
0312	00	NOP	; No operación
0313	34	INC (HL)	; Cambiar el desplazamiento
0314	15	DEC D	; Actualizar el contador de columnas
0315	20 F8	JR NZ,COL	; Si no es cero, sumar más
0317	FD 77 06	LD (IY+06H),A	; Guardar la suma en la columna 7
131A	FD 09	ADD IY,BC	; Colocar IY para la próxima fila
031C	1D	DEC E	; Actualizar el contador de filas
031D	20 EA	JR NZ,ROW	; Si no es cero, procesar la próxima fila
031F	FF	RST 38H	; Si es cero, devolver el control al ; sistema operativo.

Programa N.° 17

Posición de memoria	Código objeto	Código fuente
0320	01 07 00	LD BC,0007H
0323	1E 06	LD E,06H
0325	FD 21 80 03	LD IY,0380H
0329	3E 00	ROW: LD A,00H
032B	FD 86 00	ADD A,(IY)
032E	FD 86 01	ADD A,(IY+01H)
0331	FD 86 02	ADD A,(IY+02H)

```
0334    FD 86 03          ADD A,(IY+03H)
0337    FD 86 04          ADD A,(IY+04H)
033A    FD 86 05          ADD A,(IY+05H)
033D    FD 77 06          LD (IY+06H),A
0340    FD 09             ADD IY,BC
0342    1D                DEC E
0343    20 E4             JR NZ,ROW
0345    FF                RST 38H
```

Programa N.° 18

```
0360    1E 06                   LD E,06H
0362    FD 21 80 03             LD IY,0380H
0366    3E 00             ROW:  LD A,00H
0368    16 06                   LD D,06H
036A    FD 86 00          COL:  ADD (IY)
036D    FD 23                   INC IY
036F    15                      DEC D
0370    20 F8                   JR NZ,COL
0372    1D                      DEC E
0373    FD 77 00                LD (IY+00H),A
0376    FD 23                   INC IY
0378    20 EC                   JR NZ,ROW
037A    FF                      RST 38H
```

Paso 1

Primero mire todos los programas anteriores y observè que todos ellos realizan exactamente la misma tarea. Hay una tabla almacenada en la posición de memoria 0380 con 6 filas y 6 columnas. Cada uno de estos programas calcula el total de una línea sumando al acumulador los bytes de columnas sucesivas para una determinada fila. Estos programas varían en cuanto a sus necesidades de memoria y de tiempo debido a sus diferentes técnicas. Vamos a discutir cada programa en detalle.

Para todos los tres programas, la estructura completa del programa puede ser representada por el diagrama de flujo de la figura 7-6. Los programas difieren en los métodos elegidos para implementar los recuadros del diagrama de flujo que han sido marcados mediante un asterisco.

Programa N.° 16

El algoritmo utilizado por este programa modifica el byte de desplazamiento en la instrucción ADD A,(IY + d). Más específicamente, el par de registros HL es cargado con la posición de memoria del tercer byte de esta instrucción, con el desplazamiento. Primeramente el desplazamiento es inicializado a cero y el acumulador

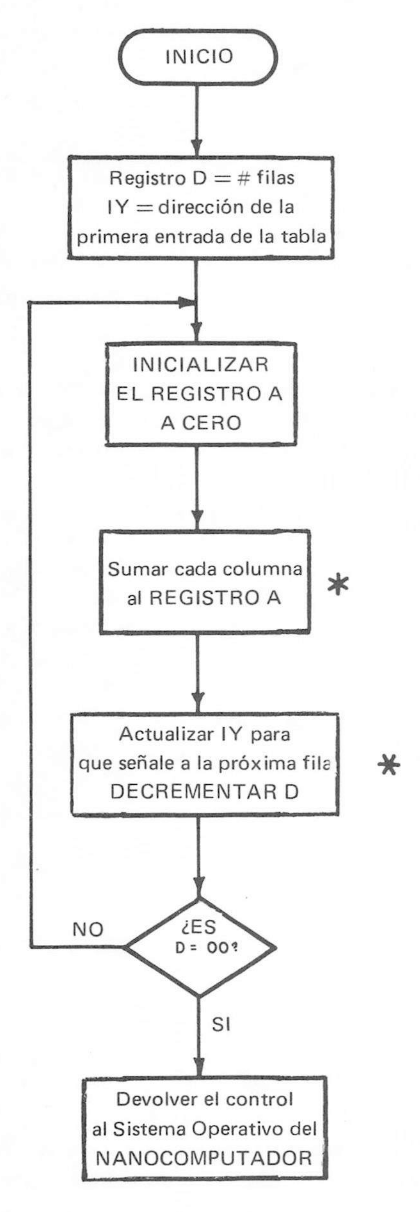

INICIO
Registro D = # filas
IY = dirección de la
primera entrada de la tabla
INICIALIZAR
EL REGISTRO A
A CERO
Sumar cada columna
al REGISTRO A
*
Actualizar IY para
que señale a la próxima fila
DECREMENTAR D
*
NO
¿ES
D = 00?
SI
Devolver el control
al Sistema Operativo del
NANOCOMPUTADOR

Figura 7-6.

se coloca a cero. A medida de que el contador de columnas cuenta el número de columnas procesadas, (IY + d) se suma al acumulador, y el desplazamiento es incrementado, INC (HL). Una vez que se han sumado todas las columnas, el contador de columnas ha sido decrementado a cero. Esto es detectado por medio de la instrucción JR NZ la cual provoca un salto a las instrucciones que procesarán a próxima fila.

Tal y como hemos mencionado, este programa se modifica a sí mismo. Las cuatro instruciones que hacen esto son:

```
LD HL,0311H
LD (HL),00H
ADD A,(IY+d)   en donde d está en la posición 0311
INC (HL)
```

Dos de estas instrucciones cambian el programa. Aquí el programa es tratado como sus propios datos. Esta es ciertamente una técnica que estimula la imaginación. Los programas pueden escribir nuevos programas o alterarse a sí mismos, cambiando dinámicamente sus propias características. Sin embargo, se deben mencionar tres desventajas importantes de esta técnica:

a. Los programas que se automodifican son a menudo difíciles de depurar.
b. Los programas que se automodifican no se pueden ejecutar en memoria de sólo lectura.
c. Los programas que se automodifican resultan muy difíciles de cambiar. No es fácil documentar tales programas. Incluso el programador original puede tener grandes dificultades en recordar los detalles de cómo trabaja el programa.

Las técnicas de automodificación son tremendamente potentes y deben ser utilizadas con extremado cuidado.

Programa N.° 17

El algoritmo utilizado por este programa es muy directo. Las seis instrucciones ADD A,(IY + d) con d = 00, . . . , 05 documentan muy claramente lo que el programa está haciendo. Desafortunadamente esto no es eficiente con respecto al espacio. Desde luego, a medida que aumenta el número de bytes que se han de sumar, menos eficaz resulta el método. Así este método está limitado a aplicaciones en las cuales se deben referenciar pocos bytes, por ejemplo el programa del experimento N.° 1.

Programa N.° 18

Este programa utiliza un algoritmo que no utiliza bytes de desplazamientos que no son cero. Esta técnica incrementa el registro IY para cada byte sumado al acumulador. Esto funciona muy bien en esta aplicación y se obtiene el programa más corto (número de bytes). La técnica aquí utilizada oculta la forma tabular de los datos porque los trata como un conjunto dimensional con IY como índice. Una razón por la cual esta técnica funciona bien aquí es debido a que las posiciones a las que nos debemos referir están en secuencia, es decir una a la derecha de la otra en la memoria. Si las posiciones a ser sumadas fueran IY + 01H, IY + 09H, IY + + 43H, IY + 44H, y IY + 56H está claro que se debería cambiar esta técnica. En contraste, la técnica del programa número 17 funcionaría tal y como está, es decir, solamente listando las columnas a ser sumadas.

En los párrafos precedentes aludimos a varias cualidades atribuidas a técnicas de programación alternativas. He aquí un resumen:

a. *Espacio:* el número de bytes de memoria necesario para guardar el programa
b. *Tiempo:* el número de estados de la CPU necesarios para ejecutar el programa
c. *Flexibilidad:* la facilidad con la cual puede cambiarse el programa
d. *Automodificación:* si el programa se modifica a sí mismo durante la ejecución
e. *Simplicidad lógica:* la facilidad con la cual el programa puede ser leído y comprendido.

Esta no es ciertamente una lista exhaustiva, pero cada uno de estos atributos debe ser considerado cuando usted intenta determinar la mejor manera de construir un programa. Típicamente, existen alternativas porque usted no puede optimizar un atributo sin sacrificar otro. Por ejemplo, para escribir programas que economicen un espacio de memoria, a menudo se debe sacrificar la rapidez de ejecución. Este es un compromiso *tiempo-espacio.* Similarmente hemos visto que la simplicidad lógica necesita a menudo más espacio, un compromiso *simplicidad-espacio.*

Paso 2

Cargar el programa número 16 y ejecutarlo en modo paso a paso, mirando con especial atención a la posición de memoria 0311, el byte de desplazamiento de la instrucción ADD A,(IY + d). Utilice la siguiente tabla para los datos:

Memoria	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	
0380	01	02	03	04	05	06	X	FILA 1
0387	02	02	02	02	02	02	X	FILA 2
038E	01	03	01	03	01	03	X	FILA 3

0395	03	03	03	03	03	03	X	FILA 4
039C	08	08	01	01	08	08	X	FILA 5
03A3	04	04	04	04	08	08	X	FILA 6

NOTA: X significa que este byte está calculado por el programa.

Paso 3

Examine cuidadosamente el programa N.° 16 para ver si existe un cambio con el cual se puede hacer el programa más corto. Una mejora que vemos es reemplazar todas las referencias a IY con HL. En general, la instrucción análoga con HL es un byte más corta.

Esperamos que usted haya visto en este experimento como la programación tiene tanto de arte como de ciencia. Una determinada tarea puede ser realizada por muchos diferentes conjuntos de instrucciones, con algunos de una forma más eficiente con respecto a otros, con respecto al tiempo, o espacio, o simplicidad lógica, o muchos otros atributos. El arte estriba en tomarse el trabajo de analizar los pros y los contras de cada alternativa, estudiando los compromisos para llegar a la mejor solución. Es normal para un programador el que escriba un programa tres o cuatro veces, si sus especificaciones merecen tal perfección. Por ejemplo una restricción típica es el espacio de memoria. Una memoria EPROM 2708 contendrá exactamente 1024 bytes. Un programa puede tener que ser escrito muchas veces para utilizar estos 1024 bytes más eficazmente.

EXPERIMENTO N.° 3

Propósito

El propósito de este experimento es el de demostrar la utilización de las operaciones del stack PUSH y POP y las instrucciones de intercambio.

Programa N.° 19

Posición de memoria	Código objeto	Código fuente	Comentarios
0130		PUSH AF	; AF primero del stack
		PUSH BC	; BC primero del stack
		PUSH DE	; DE primero del stack
		PUSH HL	; HL primero del stack
		EX, AF,AF'	; Intercambiar AF y AF'

```
EXX        ; Intercambiar pares de registros
POP HL     ; Parte alta del stack a HL
POP DE     ; Parte alta del stack a DE
POP BC     ; Parte alta del stack a BC
POP AF     ; Parte alta del stack a AF
RST 38H    ; Devolver el control al sistema
           ; operativo.
```

Paso 1

Para este programa, le pediremos que realice el ensamblado a mano. Así, se le dará una dirección de inicio y el programa fuente que es el que determina el código objeto. A continuación está un listado de la memoria para que lo utilice para comprobar su ensamblado manual:

```
0130    F5  C5  D5  E5  08  D9  E1  D1
0138    C1  F1  FF
```

Paso 2

Cargar el código objeto empezando en 0130 y verificar que sea correcto.

Paso 3

El próximo paso es colocar el stack en memoria de lectura/escritura. Tenemos dos elecciones: Lo podemos dejar donde está ahora o lo podemos cambiar. Las siguientes instrucciones son las únicas que proporciona el Z-80 para afectar la situación del stack:

```
LD SP,HL
LD SP,IX
LD SP,IY
LD SP,nn
LD SP,(nn)
```

Se puede utilizar cualquiera de estas instrucciones al principio del programa anterior para situar el stack debido a que el indicador de stack, SP, contiene la posición de memoria de lo alto del stack. Para muchos programas, una vez establecido, el registro del SP se actualiza solamente como resultado de las instrucciones PUSH y POP.

Vamos a utilizar una bonita capacidad del sistema operativo del Nanocomputador y colocar el indicador de stack en 0150 posicionando la lámpara selectora en 0150 y guardando (ST) 0150 (figura 7-7).

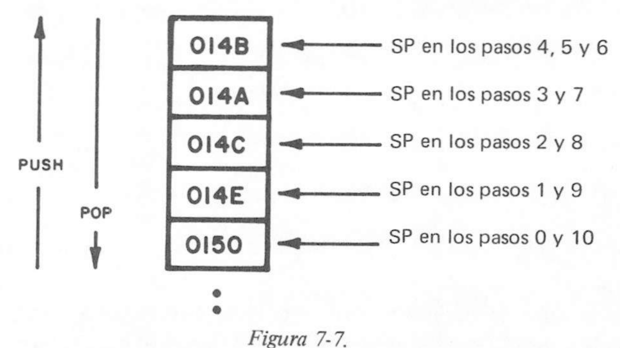


Figura 7-7.

Paso 4

Ejecute el programa en modo paso a paso con la lámpara selectora en la posición SP. Observe como el stack crece en primer lugar desde 0 a 8 bytes y a continuación disminuye de nuevo:

Paso 5

Para ayudar a controlar cuales son los registros que se guardan y en que orden, inicialice los registros de la siguiente forma:

A=01	D=05
F=02	E=06
B=03	H=07
C=04	L=08

Ejecute el programa en modo paso a paso hasta que se haya ejecutado la última instrucción PUSH, es decir hasta que el contador de programa PC = 0134. Verificar que los pares de registros han sido introducidos dentro del stack de la siguiente forma:

0148	L=08	SP
	H=07	
014A	E=06	
	D=05	
014C	C=04	
	B=03	
014E	F=02	
	A=01	
0150		

El byte HI ocupa la posición de memoria con la dirección más alta.

Paso 6

La próxima instrucción a ser ejecutada es

08 EX AF,AF' (AF'=A' y F')

Esta instrucción intercambia el contenido de estos dos pares de registros. Antes de ejecutar esta instrucción escriba el contenido de estos pares de registros:

AF=0102 AF'= (observamos 0044)

Recuerde que AF' se puede observar utilizando la tecla ARS. Cuando la lamparita ARS está encendida, salen al display los registros alternos. Pulse la tecla paso a paso una vez para ejecutar la instrucción EX AF,AF'. Entonces

AF= (observamos 0044) ;AF'=0102

La próxima instrucción es EXX que intercambia BC con BC', DE con DE', HL con HL'. Escribir el contenido de todos estos registros primero antes y a continuación después de ejecutarse EXX:

			Nuestra observación
Antes	BC=0304	BC'=	FFFF
	DE=0506	DE'=	FFFF
	HL=0708	HL'=	FFFF

Pulsar la tecla SS.

		Nuestra observación	
Después	BC=	FFFF	BC'=0304
	DE=	FFFF	DE'=0506
	HL=	FFFF	HL'=0708

Paso 7

Las próximas cuatro instrucciones son todas instrucciones POP las cuales cargan los dos primeros bytes del stack dentro del par de registros especificado. Observe el par de registros HL a medida de que usted pulsa la tecla SS para ejecutar POP HL. El byte 08 en la posición de memoria 0148 es cargado en el registro L y el 07 en la posición de memoria 0149 es cargado en el registro H.

Similarmente las próximas dos instrucciones POP cargan los pares de registros DE y BC con 0304 y 0102 respectivamente. Así, vemos que la ejecución de este

programa deja cada par de registros cargado con el mismo contenido que su par de registros alterno.

Paso 8

Deben quedar claros dos hechos importantes acerca de las operaciones del stack:

1. PUSH y POP *siempre* mueven 16 bits o dos bytes de información. Instrucciones como POP C y PUSH F no existen.
2. El orden en el cual los pares de registros son introducidos (PUSH) en el stack es opuesto al orden en el cual los pares de registros deben ser extraidos (POP) si el contenido de los registros debe ser restablecido a sus valores originales. Específicamente

```
PUSH HL
PUSH DE
POP HL
POP DE
```

es una secuencia de instrucciones equivalente a la instrucción

```
EX DE,HL.
```

Mientras que,

```
PUSH HL
PUSH DE
```

(Cualquier secuencia de instrucciones, incluyendo instrucciones que alteren DE y HL)

```
POP DE
POP HL
```

preservan los registros DE y HL tal como estaban antes de que se ejecutara la secuencia de instrucciones. Esta última secuencia de instrucciones es muy útil como usted verá más adelante cuando estudie las subrutinas.

8

Saltos, llamadas y retornos

Las instrucciones de salto, llamada y retorno comprenden la clase de instrucciones del Z-80 llamadas instrucciones de *bifurcación.* Todas ellas provocan que el flujo de las instrucciones del programa sea transferido a otros lugares de la memoria, distintos de los que normalmente habrían ocurrido si no hubiera aparecido la instrucción de bifurcación. En este capítulo, usted ampliará sus conocimientos de estas instrucciones más allá de las más simples que se han visto hasta ahora, como son, JP y JR. En particular, usted aprenderá la técnica de utilizar subrutinas. Las instrucciones de salto, llamada y retorno aparecen en la tabla 8-1.

OBJETIVOS

Después de completar este capítulo, usted será capaz de:

- Definir la *transferencia del control del programa* en términos de lo que sucede con el registro contador de programa (PC).
- Definir los indicadores de *cero, arrastre, paridad/sobrepasamiento* y *signo.*
- Definir las *llamadas a subrutina* y *retorno* con referencia a lo que sucede con el registro contador de programa (PC), el indicador de stack (SP), y el stack.
- Definir y utilizar las *instrucciones de restart.*

TRANSFERENCIAS DEL CONTROL DEL PROGRAMA

Por ahora, usted está consciente de que un programa es solamente un conjunto de bytes de memoria que representan instrucciones y datos. Normalmente estas

Tabla 8-1. Grupo JUMP (salto), CALL (llamada) y retorno

			CONDICION									
			IN-CONDIC	ARRASTRE	NO ARRASTRE	CERO	NO CERO	PARIDAD PAR	PARIDAD IMPAR	SIGNO NEG	SIGNO POSIT	REG B≠0
JUMP 'JP'	INMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVO	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	INMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENTAR B, SALTO SI NO CERO 'DJNZ'	RELATIVO	PC+e										10 e-2
RETORNO 'RET'	REGISTRO INDIRECTO	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETORNO DE INT 'RETI'	REGISTRO INDIRECTO	(SP) (SP+1)	ED 4D									
RETORNO DE INT NO ENMASCARABLE 'RETN'	REGISTRO INDIRECTO	(SP) (SP+1)	ED 45									

NOTA: CIERTOS INDICADORES TIENEN MAS DE UNA APLICACION REFERIRSE A LA SECCION 6.0 PARA DETALLES

Cortesía Zilog, Inc.

instrucciones se ejecutan secuencialmente, es decir una a continuación de otra, hasta que algo (como una instrucción JP) cambia este modo de ejecución. Vamos a examinar en detalle como ejecuta el Z-80 un programa guardado en alguna parte de la memoria.

La CPU del Z-80 guarda la dirección de la próxima instrucción que va a ser ejecutada en el registro PC (contador de programa). Así la ejecución de un programa consiste en repetir los siguientes pasos hasta que se ejecuta una instrucción de ALTO (HALT):

Paso 1. Leer (PC) en un registro de instrucción que está dentro del chip del Z-80. Recuerde que la notación (PC) significa el contenido de la posición de memoria direccionada por el contador de programa de 16 bits. (PC) es el primer byte de la instrucción, y así, es un código de operación que empezará a definir la instrucción a ser ejecutada. En algunos casos, este byte será la instrucción completa. En otros casos la CPU tendrá que leer otro byte antes de que conozca, qué longitud tiene la instrucción en bytes.

Paso 2. Decodificar el primer byte de la instrucción, y determinar si es necesario leer bytes adicionales. Incrementar la dirección en el contador de programa de forma que señale a la próxima posición de memoria, es decir el próximo byte del programa. Si la decodificación indica que la instrucción solamente es de un byte, entonces ir al paso 4.

Paso 3. Continuar leyendo (PC) e incrementar el PC hasta que se haya leído toda la instrucción, en total un máximo de cuatro bytes.

Paso 4. Ejecutar la instrucción y volver entonces al paso 1.

En el momento en que se debe empezar la ejecución de una instrucción el registro PC de 16 bits siempre guarda la dirección de memoria del primer byte de la próxima instrucción. Como usted podrá ver, todas las instrucciones de bifurcación, como los saltos, llamadas y retornos, actúan directamente en el registro PC para cambiar la secuencia en la cual son leídas las posiciones de memoria y ejecutadas como instrucciones.

INSTRUCCIONES DE SALTO INCONDICIONAL

Las instrucciones de salto hacen que el control del programa sea transferido a una dirección especificada por la misma instrucción. Después que se ha leído el primer byte de la instrucción y se ha decodificado mediante la CPU del Z-80, el tipo de instrucción ha quedado completamente determinado. Entonces se lleva a cabo la ejecución de la instrucción leyendo el próximo byte o bytes para determinar la dirección del salto, después de lo cual la dirección del salto es cargada en el registro PC. Cuando la CPU empieza el próximo *ciclo de instrucción,* el contador de programa contiene la dirección de la instrucción direccionada por el byte(s) de dirección de la instrucción previa de salto. Ilustraremos el efecto de una instrucción JP en el PC a continuación:

Secuencia de direcciones	Instrucción	(PC = 1000 inicialmente)
1000	LD A	
1001	00H	

→1002	INC A	PC = 1002 después de ejecutar LD A,00H
1003	JP	PC = 1003 después de ejecutar INC A
1004	02H	
1005	10	
		PC = 1002 después de la ejecución de JP 1002H. (Una instrucción de tres bytes que no fuera de salto habría dejado el contador de programa PC = 1006 después de su ejecución.)

Obsérvese que la instrucción JP afecta solamente al registro PC. No se efectúa ninguna otra operación. En el caso de una instrucción JP, la dirección completa del salto está contenida en los bytes segundo y tercero de la instrucción. Así, la ejecución de la instrucción de salto consiste en cargar los dos últimos bytes de la instrucción en el registro PC. El efecto de una instrucción de salto en el control de un programa es ilustrado en la figura 8-1.

Los saltos pueden ir "hacia adelante" o "hacia atrás". El flujo del programa solamente sigue los cambios del registro contador de programa de una posición a otra. Generalmente, cuanto mayor sea el número de saltos en un programa más difícil será cambiarlo y depurarlo.

Existen dos clases de instrucciones de salto implementadas en el microprocesador Z-80. *Saltos absolutos* y *saltos relativos.* La instrucción JP utilizada en el ejemplo anterior es un salto absoluto porque la dirección de salto de dos bytes está especificada como parte de la instrucción. Los saltos relativos (JR) especifican un desplazamiento en complemento a dos en un byte de la instrucción. Así, la ejecución de un salto relativo incluye un paso extra para determinar la dirección del salto a partir de la suma del valor del PC actual y del byte de desplazamiento. A continuación damos el mismo programa anterior, pero en este caso utilizamos un salto relativo.

Dirección	Instrucciones	(PC = 1000, inicialmente)
1000	LD A,	
1001	00H	
→1002	INC A	PC = 1002 después de la ejecución de LD A,00H
1003	JR	PC = 1003 después de la ejecución de INC A
1004	FDH	
		PC = 1002 después de la ejecución de JR FDH. (Una instrucción de dos bytes que no fuera de salto habría dejado el PC = 1005 después de su ejecución.)

Observe que FD es la representación en complemento a dos de –3. En las instrucciones de salto relativo, el byte de desplazamiento da el número de bytes antes (negativo) o después (positivo) de la posición indicada por un *contador de programa actualizado normalmente.* En el ejemplo anterior, el contador de programa normal-

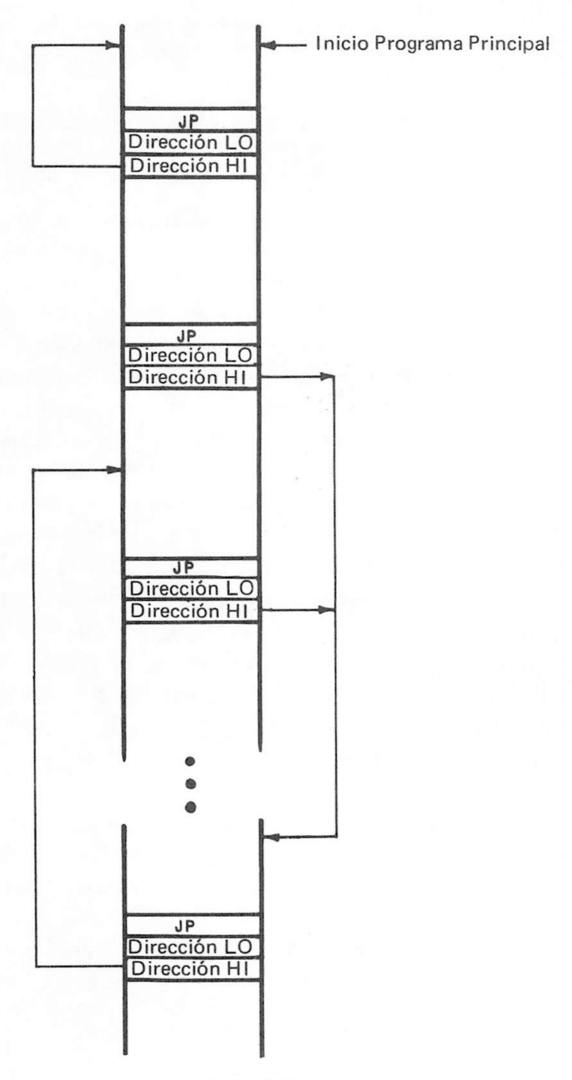


Figura 8-1.

mente habría sido 1005 después de la ejecución de una instrucción de dos bytes que no fuera de salto. Tres bytes ANTES de 1005 está la posición 1002, la dirección de salto, así el desplazamiento es –3 o FD. Una equivocación común que cometen los programadores consiste en determinar el byte de desplazamiento referente al PC ANTES de la ejecución de la instrucción JR. Esto es incorrecto e inva-

riablemente produce conflictos. *El desplazamiento para una instrucción de salto relativo debe estar siempre determinado con relación a la primera posición después de la instrucción de dos bytes JR.*

Vamos a escribir algunas diferencias entre los saltos absolutos y relativos:

1. La instrucción JP utiliza tres bytes por instrucción, un byte de código de operación más una dirección de dos bytes. Un JR utiliza dos bytes por instrucción, un byte para el código de operación más un byte para el desplazamiento.
2. La instrucción JP provoca que el control del programa se transfiera a cualquier posición de la memoria. La instrucción JR puede hacer que el control del programa se transfiera a posiciones de la memoria comprendidas en la gama de −128 a +127 bytes de la posición de memoria situada después del byte de desplazamiento. Esta es la limitación de los desplazamientos de un byte.
3. Las instrucciones JP generalmente no son *reubicables* mientras que las instrucciones JR lo son. Un atributo muy importante de la instrucción JR es que está referida solamente a la posición de memoria de la instrucción. Esto implica que la instrucción JR mantiene su integridad independientemente de su posición *absoluta* de memoria. Cualquier programador que ha tenido que mover una gran parte de un programa hacia atrás de un byte en la memoria para incluir una nueva instrucción puede apreciar la conveniencia de escribir código reubicable (independiente de su posición en la memoria) (suponiendo que se efectúa el ensamblado a mano).

Antes de que consideremos los saltos condicionales deseamos destacar que existen tres saltos absolutos que utilizan direccionamiento por registro indirecto. Esto es, estos saltos, JP (HL), JP (IX) y JP (IY) indican la dirección del salto como el contenido de uno de los pares de registros HL, IX o IY. Estas, así como también todas las instrucciones de bifurcación del Z-80, están contenidas en la tabla 8-1.

INDICADORES Y SALTOS CONDICIONALES

Las instrucciones JP y JR son ambas saltos incondicionales. Esto significa que el control del programa *siempre* es transferido a la dirección absoluta o relativa cuando se ejecuta la instrucción. Los saltos condicionales son instrucciones que transfieren el control de programa a una posición diferente *dependiendo* de alguna condición que debe ser alcanzada. Por ejemplo, la instrucción JP NZ provoca un salto si el indicador de cero está al nivel lógico 0. (Aunque piense que NZ, no cero, y cero lógico, parecen contradictorios, realmente no lo son, como usted verá.)

Cuando introducimos esta instrucción en el capítulo 6, no discutimos los registros indicadores. Lo haremos a continuación.

La CPU del Z-80 contiene dos registros de indicadores F y F', uno para cada de los dos conjuntos de registros de uso general. Cada registro de indicadores contiene seis *flags,* o bits de información, que quedan afectados individualmente por varias instrucciones del Z-80. Decimos que un indicador está a 1 si su valor es el valor lógico 1, y a cero si su valor es el cero lógico. Cuatro de los seis indicadores son utilizados como condiciones para las instrucciones de salto, llamada y retorno. Estos son:

1. *INDICADOR DE ARRASTRE* (*C*): Este indicador está afectado por las instrucciones de suma, resta, rotación y desplazamiento. Durante una operación de suma, se coloca a 1 si se produce un arrastre en el bit más significativo del acumulador. Durante una operación de resta, el indicador de arrastre se coloca a 1 si se produce un arrastre para la resta en el bit más significativo del acumulador. Muchas operaciones de rotación y desplazamiento añaden el indicador de arrastre como un noveno bit a manipular. Introduciremos y discutiremos a fondo las instrucciones aritméticas y lógicas en capítulos subsiguientes.

Dos instrucciones manipulan directamente el indicador de arrastre:

SCF: (set carry flag) obliga al bit C que es el indicador de arrastre a tomar el valor lógico 1.

CCF: (complement carry flag) cambia el nivel lógico actual del bit C. Si C = 1, CCF pone a cero el bit C. Si C = 0, CCF coloca a 1 el bit C.

2. *INDICADOR DE CERO* (*Z*): El estado de este indicador queda afectado por muchas instrucciones. Las operaciones que cambian al acumulador normalmente afectan al indicador de cero, colocándolo a 1 si el acumulador es cero, y colocándole a cero si el acumulador no es cero. Las instrucciones BIT colocan a 1 el indicador de cero si el bit especificado es cero y lo colocan a cero si el bit vale 1. Las instrucciones de comparación, CP, comprueban si existe igualdad entre el byte especificado y el acumulador. Si existe igualdad, el indicador de cero se coloca a 1; si no a cero. Más tarde se discutirán otras instrucciones que afectan el indicador de cero. El indicador de cero es el único para el cual existe alguna inconsistencia y confusión con respecto a las condiciones en las cuales el indicador es cero o uno. La razón para ello es la forma en la cual está manipulado el indicador: si el indicador de cero vale cero significa que el resultado de la operación no fue cero. Así cuando aparece la instrucción JP NZ en un programa, se producirá salto si el *resultado no*

es cero; o en términos del indicador de cero, salto si el *indicador de cero* (*Z*) *es cero.* Si el indicador de cero es igual a cero significa que el resultado de la operación previa no fue cero. Esto se presta a confusión. Nuestro mejor consejo es sugerir que usted memorice este hecho y que piense cuidadosamente cuando utilice el indicador de cero como una condición para saltar. El hecho importante es que la condición NZ se refiere al resultado de la operación previa, y *no* al indicador.

3. *INDICADOR DE SIGNO* (*S*): Este indicador es una copia del bit más significativo del resultado de una operación. Los resultados se guardan normalmente, pero no siempre en el acumulador. El propósito del indicador S es el de señalar que el resultado en complemento a dos es positivo o negativo. Así el indicador S se coloca a 1 (= 1) si es negativo y se coloca a cero (= 0) para resultados positivos.

4. *INDICADOR DE PARIDAD/SOBREPASAMIENTO* (*P/V*): El indicador P/V tiene dos utilidades:
   (a) Indicar la paridad del resultado de una instrucción lógica, de rotación, desplazamiento o instrucción de entrada.
   (b) Indicar sobrepasamiento como resultado de una operación aritmética en complemento a dos.

La palabra *paridad* se refiere al número de bits en un byte que están al nivel lógico 1. Si el número de bits que están a 1 es impar, entonces se dice que el byte tiene una *paridad impar.* Si el número de bits que están a 1 es par decimos que el byte tiene *paridad par.* Por ejemplo, la paridad de FF es par, y la paridad de 01 es impar.

El indicador de paridad se *coloca* a 1 si el resultado es *par* y a cero si la paridad del resultado es *impar.*

En el capítulo 6, discutimos la detección del sobrepasamiento en la aritmética en complemento a dos. Si la suma de dos números positivos en complemento a dos resulta en un número negativo en complemento a dos, el indicador de sobrepasamiento se *coloca* a 1. Similarmente el indicador se *coloca* a 1 si la suma de dos números negativos nos da un resultado positivo. El indicador V se *pone* a cero si no ocurre un sobrepasamiento.

Es importante entender la diferencia entre el *indicador de arrastre* (*C*) y el *indicador de sobrepasamiento* (*V*). Si se suman o restan dos números binarios, el indicador C se utiliza para detectar cualquier sobrepasamiento. Si se suman o restan dos números en complemento a dos, el indicador V se utiliza para detectar cualquier sobrepasamiento que se produzca. Estos dos indicadores *no* son intercambiables. Aquí está un ejemplo que lo prueba.

	1 1 1 1 1 0 1 1	es la representación en complemento a dos para –5
	+1 1 1 1 0 0 0 0	es la representación en complemento a dos de –16
SUMA:	1 1 1 0 1 0 1 1	es la representación en complemento a dos de –21

C = 1 porque hay un arrastre del bit más significativo
V = 0 porque –21 es correcto; no existe sobrepasamiento.

5. *INDICADORES DE MEDIO ARRASTRE (H) Y DE RESTA (N)*: Los dos últimos indicadores son altamente especializados y se utilizan solamente en la aritmética basada en un esquema de codificación binaria llamado *binario codificado en decimal (BCD)*. En este momento solamente diremos que los dos indicadores se llaman el *medio arrastre (H)* y el *indicador de resta (N)*. Los dos son importantes para la instrucción *ajuste decimal del acumulador (DAA)*. Ninguno de los indicadores H o N es utilizado en conexión con las *bifurcaciones* o saltos, como lo son los primeros cuatro indicadores, es decir, no se pueden "testar", con ninguna de las instrucciones de bifurcación condicional. El formato del registro de indicadores es:

S	Z	X	H	X	P/V	N	C

en donde X significa que el valor del bit no tiene interés. La X algunas veces es llamada una condición de "no importa" porque nunca se le presta ninguna atención. La carta de programación de la SGS-ATES de la CPU del Z-80, así como la tabla en el apéndice I, resume todas las instrucciones del Z-80 que afectan a los indicadores.

Ya hemos discutido el salto condicional JP NZ, el cual transfiere el control del programa a la drección especificada dependiendo de que el indicador de cero (Z) esté a cero. Para cada uno de los indicadores S, Z, P/V y C existe una instrucción de salto condicional JP que depende de si el indicador correspondiente está a uno o a cero:

JP NZ: Salto si el indicador de cero está a cero (resultado no cero)
JP Z: Salto si el indicador de cero (Z) está a uno (resultado cero)
JP NC: Salto si el indicador C está a cero (no arrastre)
JP C: Salto si el indicador C está a uno (arrastre)
JP PO: Salto si el indicador de paridad (P) está a cero (paridad impar o sin sobrepasamiento en la operación previa)
JP PE: Salto si el indicador P está a uno (paridad par o sobrepasamiento en la operación previa)
JP P: Salto si el indicador de signo S está a cero (resultado positivo)
JP M: Salto si el indicador S está a cero (resultado "negativo")

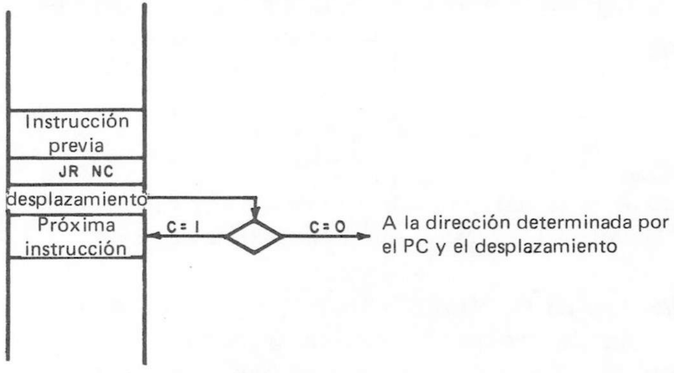


Figura 8-2A.

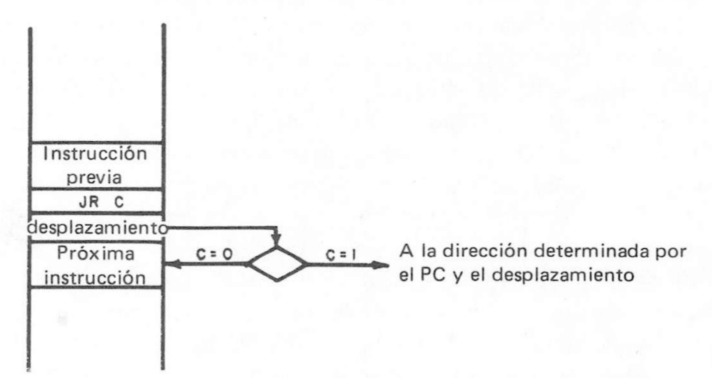


Figura 8-2B.

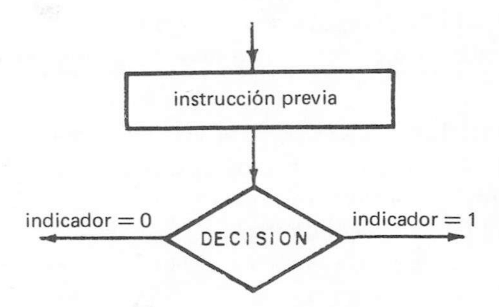


Figura 8-2C.

Existen cuatro saltos condicionales relativos –JR NZ, JR Z, JR NC, JR C– además de los saltos relativos incondicionales, JR.

Vamos a estudiar los saltos relativos cuyas acciones dependen del estado del in-

dicador de arrastre (C). Los efectos de estas instrucciones JR NC y JR C están expresados en los diagramas de las figuras 8-2A y 8-2B. En la figura 8-2C, el símbolo de decisión que es utilizado corrientemente en el diagrama de bloques del programa, indica que lo que sucede a continuación depende del estado de un indicador. Más adelante en este capítulo, y en capítulos sucesivos, usted verá muchos ejemplos de programas que utilizan saltos condicionales.

Existe una instrucción de salto relativo condicional muy especializada cuyo mnemónico es DJNZ. La instrucción se ejecuta en tres pasos:

a. El registro B se decrementa (de 1).
b. Se efectúa una comprobación para ver si el contenido del registro B es 00.
c. Si B no es 00, entonces tiene lugar el salto relativo. De lo contrario, se ejecuta la próxima instrucción en secuencia.

El diagrama de flujo de la figura 8-3 muestra como trabaja la instrucción DJNZ. Muy a menudo en la programación, se necesita ejecutar el mismo grupo de instrucciones, un determinado número de veces. La instrucción DJNZ está diseñada con este propósito. La figura 8-4 muestra como se puede utilizar la instrucción DJNZ. Como usted puede ver esta instrucción es muy útil.

LLAMADAS Y RETORNOS

Una instrucción de *llamada* es una instrucción de salto que "recuerda" desde donde dió el salto. La instrucción de *retorno* hace que la CPU continúe la ejecución del programa en la instrucción siguiente a la última llamada ejecutada. Ver figura 8-5.

Como usted puede imaginar, las instrucciones CALL (llamada) y RET (retorno) son muy útiles porque permiten que el mismo bloque de código sea ejecutado por medio de la utilización de instrucciones de llamada desde lugares muy diferentes en un programa. Una de las utilizaciones normales de las subrutinas es para los bucles de retardo en un programa. Una rutina de retardo puede ser escrita para un cierto período de tiempo, por ejemplo un milisegundo. Cada vez que el programa necesita un retardo de n milisegundos, la subrutina es llamada n veces. Los fabricantes de hardware, los que desarrollan software, y los varios grupos de usuarios de computadores tienen colecciones de subrutinas útiles para que puedan ser utilizadas por los programadores en sus computadores. Estas colecciones son llamadas *librerías.* Estas librerías pueden incluir subrutinas de multiplicación y división, subrutinas para cálculos trigonométricos, logarítmicos y funciones exponenciales, y otras tareas especializadas.

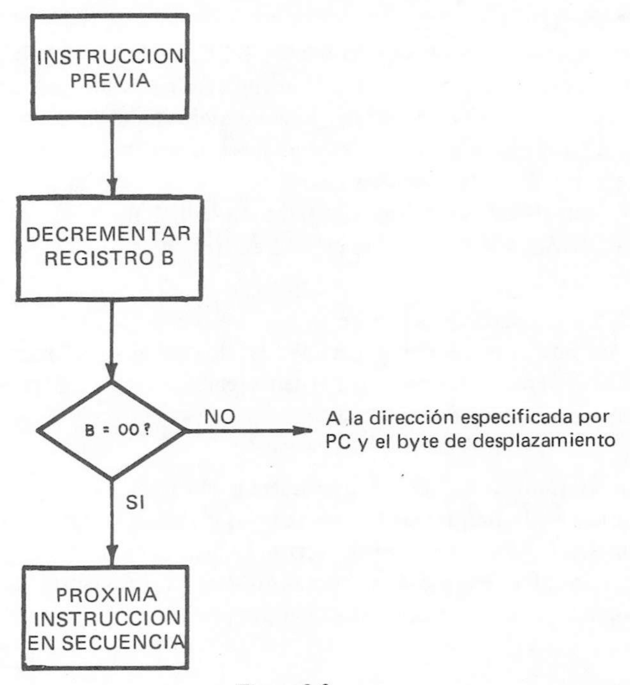


Figura 8-3.

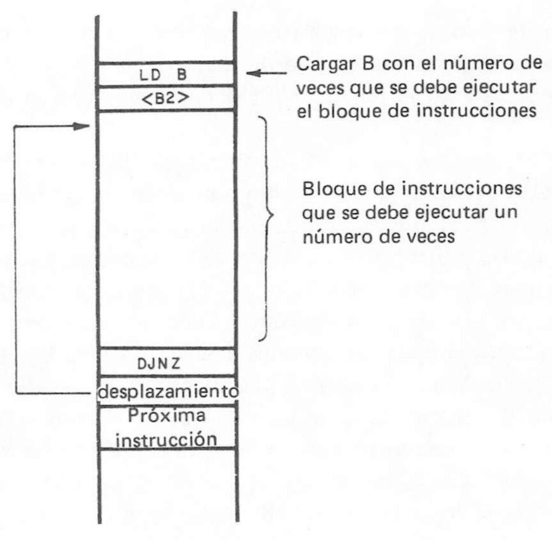


Figura 8-4.

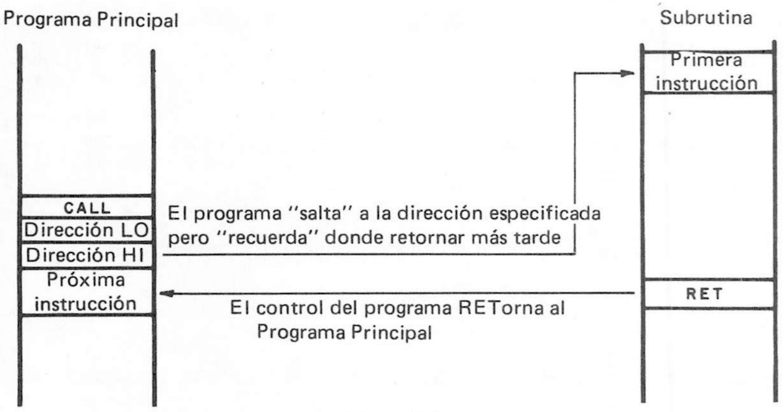


Figura 8-5.

Ahora que usted conoce lo *que* hacen las instrucciones CALL y RET debe aprender *como* lo hacen. En primer lugar la instrucción CALL. Tal y como en la instrucción JP, la instrucción CALL especifica una posición de memoria en los bytes segundo y tercero de la instrucción. Estos dos bytes son cargados en el registro PC para efectuar la transferencia del control del programa. Sin embargo, antes de que el PC sea cambiado, la instrucción CALL realiza un primer paso que lo distingue de una instrucción JP. Para recordar donde debe volver en el programa principal, el PC es introducido en el stack –en primer lugar el byte HI, y a continuación el byte LO– *mientras que el PC todavía está señalando a la próxima instrucción después de la llamada de tres bytes.*

La subrutina mantiene el control hasta que se encuentra una instrucción de retorno. El retorno incondicional, RET, hace que el PC se cargue con los dos bytes que están primeros en el stack; el primer byte se convierte en el byte LO, y el próximo byte del stack se convierte en el byte de dirección HI. Si el stack ha sido utilizado adecuadamente, el RET debe producirse cuando los dos bytes que están los primeros en el stack son los que fueron introducidos por la llamada previa. Así el control del programa retorna a la primera instrucción después de la llamada. Para ilustrar este proceso vamos a estudiarlo paso a paso pasando por un CALL y entonces RET de una subrutina.

Dirección				
0300	LD SP	Inicio del programa: inicializa el indicador de stack	SP=	0400 [1]
0301	00H	en 0400	PC=	0303
0302	04			
0303	CALL	Llama la subrutina en 0310: introduce 0306 en el	SP=	03FE [2]
0304	10H	stack; decrementa el indicador de stack de 2; carga	PC=	0310
0305	03	PC con 0310		
0306	HALT	Alto		

0310	INC A	Inicio de la subrutina	SP=	03FE
0311	INC B		PC=	0311
0312	INC C			
0313	DEC D			
0314	DEC E		PC=	0314
0315	RET	Retorno al programa principal: extrae (POP) dos bytes fuera del stack y los introduce en el PC; incrementa SP de 2.	SP= PC=	0400 [3] 0306

STACK: El stack solamente se cambia con las siguientes instrucciones

[1] 0300 LD SP
[2] 0303 CALL
[3] 0315 RET

(Nótese que para los diagramas, las direcciones se *incrementan* hacia atrás.)

	Stack después (1)		Stack después (2)		Stack después (3)	
03FE	X		06	SP=03FE	06	
03FF	X		03		03	
0400	X	SP=0400	X		X	SP=0400
	X		X		X	
	•		•		•	
	•		•		•	
	•	PC=0303	•	PC=0310	•	PC=0306

(X es desconocido y no importa su valor)

Incluso el programador más experimentado puede caer en tres errores críticos:

1. El stack crece tanto que se expande en la memoria RAM hasta que empieza a "comerse" el programa que está utilizando, o los datos que utiliza el mismo programa; o, el stack crece tanto que intenta expansionarse dentro de la zona ocupada por memoria ROM o PROM y no puede aceptar nuevos bytes.
2. La subrutina que se ha llamado empieza a manipular el stack de tal manera que la instrucción de retorno es ejecutada con un par de bytes equivocados situados en la parte alta del stack. ¿Quién conoce dónde mandará la CPU el control del programa?
3. Usted se olvidó de inicializar el indicador de stack y empezó con el stack situado en *cualquier parte* en la memoria. En particular, *"cualquier parte"* puede ser en medio de su *programa,* o en la posición FFFF en donde no se dispone de memoria de lectura-escritura.

Cualquiera de las tres opciones anteriores del mal empleo del stack puede resultar en la ejecución de datos del stack siempre aventurada, si no con equivocaciones garantizadas!

Aun con estos riesgos, la técnica de llamadas a subrutinas, es muy útil y es una práctica que es muy interesante desarrollar. Raramente se puede practicar demasiado. Los programas que están estructurados utilizando las subrutinas son normalmente más fáciles de entender, más fáciles de cambiar, y también de depurar, que si no se hubieran utilizado subrutinas. El experimento número 2 investiga cuando se deben utilizar subrutinas.

Similarmente a la instrucción JP, las instrucciones CALL y RET tienen equivalentes condicionales que realizan una llamada o retorno dependientes del estado de

Tabla 8-2. Grupo Restart

DIRECCION DE LLAMADA	CODIGO DE OPERACION	
0000_H	C7	'RST 0'
0008_H	CF	'RST 8'
0010_H	D7	'RST 16'
0018_H	DF	'RST 24'
0020_H	E7	'RST 32'
0028_H	EF	'RST 40'
0030_H	F7	'RST 48'
0038_H	FF	'RST 56'

Cortesía Zilog, Inc.

un indicador. Todos ellos aparecen en la tabla 8-1. La tabla 8-2 muestra las instrucciones de restart, RST N. Estas son instrucciones de un byte que realizan una llamada a una dirección especificada en el mismo código de operación. En otras palabras, las instrucciones RST son llamadas incondicionales de un byte. La dirección necesaria de dos bytes en una instrucción CALL se evita mediante un direccionamiento modificado por página cero. Se pueden especificar una entre ocho direcciones hex para otras tantas subrutinas dependiendo de tres bits contenidos en el código de operación: 0000, 0008, 0010, 0018, 0020, 0028, 0030 y 0038.

INTRODUCCION A LOS EXPERIMENTOS

Este grupo de ejercicios se concentra en técnicas de programación muy importantes que utilizan las varias instrucciones de bifurcación del Z-80. Los primeros

experimentos le introducen a las instrucciones dándole programas simples para ejecutar mientras se observa muy de cerca el contador de programa (PC) así como otros registros afectados y posiciones de memoria. Los últimos experimentos discuten técnicas tales como la transferencia de parámetros a las subrutinas y la utilización de tablas de salto. Los experimentos que usted realizará se pueden resumir como sigue:

Experimento N.°	Comentarios
1	Demuestra la instrucción DJNZ para utilizarla en una rutina de bucle de retardo. También se demuestra la ejecución de puntos de paro.
2	Demuestra como convertir el programa del experimento N.° 1 a una subrutina y da un programa de llamada como ejemplo. Se discute cuando se deben utilizar subrutinas.
3	Demuestra la instrucción RST.
4	Demuestra cuatro técnicas para pasar parámetros a las subrutinas.
5	Demuestra las técnicas de utilización de las tablas de salto.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar la instrucción DJNZ utilizándola para implementar un bucle de retardo. También se demuestra la técnica para insertar, utilizar y eliminar puntos de paro durante la ejecución de un programa.

Programa N.° 20

Posición de memoria	Código objeto	Código fuente		Comentarios
0100	06 09		LD B,09H	; Inicializar registro B
0102	0E FF	LOOP1:	LD C,FFH	; Inicializar registro C
0104	16 FF	LOOP2:	LD D,FFH	; Inicializar registro D
0106	15	LOOP3:	DEC D	; bucle interior
0107	20 FD		JR NZ,LOOP3	; Decrementa el registro D

```
0109      0D        DEC C          ; Decrementa al bucle interior
010A      20 F8     JR NZ,LOOP2    ; registro C
010C      10 F4     DJNZ LOOP1     ; Decrementa bucle externo
010E                               ; registro B
          FF        RST 38H        ; Devuelve el control al sistema
                                   ; operativo del Nanocomputador
```

Paso 1

Cargar el programa anterior y ejecutarlo con diferentes valores iniciales para el registro B. Como usted puede ver en el ejemplo utilizamos 09. Observe las grandes variaciones de tiempo que se pueden obtener y durante el cual el display permanece apagado, cambiando el valor inicial del registro B.

Registro B	Tiempo total de ejecución del programa en el Experimento N.º 1
01	0,4420844 segundos (aproximadamente)
09	3,66123772 segundos (aproximadamente)
FF	105,47 segundos (aproximadamente)

La instrucción DJNZ automáticamente decrementa el registro B. Así se ahorra una instrucción, DEC B, utilizando este salto especializado.

Paso 2

Supongamos que usted desea observar que le sucede al registro B mientras que el programa anterior ejecuta la instrucción DJNZ. Usted podría avanzar paso a paso decrementando C y D, decrementando C 255 veces mientras que D se decrementa una vez. Sin embargo esto costaría mucho tiempo. La mejor alternativa consiste en insertar un punto de paro antes de que se ejecute la instrucción DJNZ. Por ejemplo, el punto de paro se podría insertar en la posición 010C, puesto que la instrucción situada en el punto de paro no se ejecuta antes de que el programa se pare. Entonces, colocando la lámpara del selector en la posición BC y apretando la tecla SS le permitirá ver como se decrementa el registro B. Pulse 00 de nuevo y la ejecución se detendrá la próxima vez que el PC tiene el mismo valor que la dirección del punto de paro, es decir cuando la instrucción DJNZ está a punto de ser ejecutada de nuevo.

Para colocar el punto de paro, colóquese en Modo BRK (Breakpoint) pulsando la tecla BRK. La lámpara BRK se iluminará y usted debe ver un solo cero en el display de datos. Pulse y mantenga hacia abajo la tecla INC. Usted debe ver que el 0 cambia a 1, 2, 3 hasta 7 y de nuevo cero. Si no hay otros puntos de paro colocados, no aparecerán otros dígitos en el display de datos o en el de direcciones. Incremente

el contador de punto de paro (el dígito que está solo en el display) para leer 3. Entre 010C y pulse LA. Usted debe ver:

0 1 0 C   3 1 0

010C es la dirección del punto de paro, 10 es el contenido de esta dirección (el código de operación de DJNZ), y 3 es el contador de punto de paro. Este punto de paro se podría haber entrado como cualquier número de punto de paro del 0 al 7. Hemos elegido 3 arbitrariamente. Salga del modo BRK, pulsando de nuevo la tecla BRK. La lamparita BRK se apagará. Ejecute el programa a toda velocidad empezando en la posición 0100. El display se apagará momentáneamente, y a continuación nos mostrará:

0 1 0 C   1 0

Esto es, el PC = 010C y la próxima instrucción que se va a ejecutar es DJNZ. Posicione la lámpara selectora en BC y pulse la tecla SS una vez. El registro B debe decrementarse de 1 y el PC debe leer 0102. Para ver como se ejecuta la instrucción DJNZ por segunda vez, pulsar GO de nuevo. La ejecución se detendrá con PC = = 010C de nuevo. Usted puede avanzar paso a paso para cada vez que se encuentra la instrucción DJNZ en la ejecución del programa de esta manera.

Paso 3

Elimine el punto de paro entrando en el modo BRK, mostrando en el display el punto de paro que se quiere borrar utilizando la tecla INC y pulsando finalmente GO.

EXPERIMENTO N.º 2

Propósito

El propósito de este experimento es demostrar como se puede convertir el programa del experimento número 1 en una subrutina y dar un programa de muestra que llame a la subrutina. También se discute cuando se deben utilizar subrutinas.

Programa N.º 21

Posición de memoria	Código objeto	Código fuente	Comentarios
011B	31 FF 01	LD SP,01FFH	; Situar el stack del sistema
011E	06 03	LD B,03H	; Especificar el valor del registro B

0120	CD 02 01	CALL DELAY	; Llamar a la subrutina
0123	3E 00	LD A,00H	; Cargar el acumulador con 00
0125	FF	RST 38H	; Devolver el control al sistema operativo

Subrutina Delay

Posición de memoria	Código objeto	Código fuente	
0102	0E FF	DELAY:	LD C,FFH
0104	16 FF	LOOP2:	LD D,FFH
0106	15	LOOP3:	DEC D
0107	20 FD		JR NZ,LOOP3
0109	0D		DEC C
010A	20 F8		JR NZ,LOOP2
010C	10 F4		DJNZ DELAY
010E	C9		RET

Paso 1

El programa del experimento número 1 aparece bajo el título Subrutina Delay. Lo hemos convertido en una subrutina, entre otras cosas, reemplazando la intrucción RST 38H por la instrucción RET. Así, ahora la subrutina devuelve el control a la rutina que la ha llamado, listada como Programa 21, en lugar de saltar al sistema operativo del Nanocomputador. Se han hecho otras dos modificaciones. Primeramente la etiqueta LOOP1 se ha cambiado a DELAY. Esto se ha hecho por razones puramente estéticas puesto que pareció que era más "autodocumentado" que el programa llamara a la subrutina por el nombre DELAY en lugar de LOOP1. Así que este cambio no era totalmente necesario. El segundo cambio fue omitir la instrucción LD B, <B2> en la subrutina. Cambiamos esta instrucción al programa que llama a la subrutina de forma que el programa que la llama pueda especificar la longitud del retardo cargando el registro B un momento antes de pasar el control a la subrutina DELAY. Este no era un cambio necesario para convertir el programa del experimento número 1 en una subrutina. Sin embargo, le añade una gran flexibilidad a la forma en que se puede utilizar esta subrutina. Concretamente, esta subrutina se puede utilizar para conseguir retardos comprendidos entre 256 intervalos de tiempo distintos dependiendo del contenido del registro B antes de que ésta sea llamada. Esta técnica de hacer que el programa que llama a la subrutina especifique valores cruciales para el funcionamiento de la misma se denomina *transferencia de parámetros.* En este caso el parámetro es el contenido del registro B el que determina la duración del retardo. Existen otras formas de pasar parámetros a las subrutinas los cuales se discuten en el experimento número 4.

En conclusión, vamos a resumir los cambios necesarios para convertir un programa en una subrutina. Además, vamos también a identificar los elementos necesarios en una subrutina de llamada.

Para cambiar un programa en una subrutina:

El único cambio necesario es el de insertar instrucciones de RET condicionales o absolutas cuando el control debe retornar al programa que ha llamado a la subrutina. Puesto que la instrucción RET solamente utiliza un byte de memoria esto puede que nos ahorre espacio de memoria, por ejemplo si la instrucción RET reemplaza los dos o tres bytes de las instrucciones JR o JP. En el peor de los casos el programa se alarga de un byte para cada nueva instrucción de RET.

Para el programa que llama a la subrutina:

Es crítico especificar explícitamente donde debe residir en la memoria el stack del sistema. Esto se logra mediante la instrucción de tres bytes

LD SP,<B3><B2>

Si el programa no utiliza el stack para otras tareas, es decir que no contiene instrucciones PUSH y POP, entonces la llamada a la subrutina ha costado tres bytes de la memoria para colocar el stack. Si el programa utiliza el stack para otras cosas, entonces el stack ya se habrá fijado en otra parte del programa.

Para cada llamada a la subrutina, se necesitan normalmente tres bytes. (La instrucción RST se puede utilizar solamente para casos especiales.) Si la instrucción CALL reemplaza a un salto absoluto (JP), el programa no se hace más largo. En el peor de los casos, el programa se incrementa de tres bytes para cada llamada (CALL).

Cargar el PROGRAMA y la SUBRUTINA en las direcciones indicadas. (Gran parte de la SUBRUTINA ya ha sido cargada en el experimento número 1.)

Paso 2

Insertar un punto de paro en la posición 010E, es decir justamente antes de la instrucción RET. Ejecutar el programa principal en modo paso a paso empezando en la posición 011B. Mirar al registro SP antes y después de la ejecución de la instrucción CALL DELAY:

Antes: SP=__________
Después: SP=__________

Observamos que antes de la llamada a la subrutina SP = 01FF, y que después de la llamada a la subrutina SP = 01FD.

Paso 3

Observar los dos bytes que están en lo alto del stack, es decir en las posiciones

01FE y 01FD. Usted puede ver que las direcciones de la próxima instrucción después de la instrucción CALL DELAY ha sido introducida (PUSH) dentro del stack, es decir (SP) = (01FD) = 23 y (SP + 1) = (01FE) = 01. Obsérvese también que PC = 0102, la dirección de la primera instrucción de la subrutina DELAY.

Paso 4

Pulsar la tecla GO para continuar la ejecución del programa hasta que se encuentre la instrucción RET en la posición 010E. Compruebe los registros SP y PC antes y después de que se haya ejecutado la instrucción RET:

Antes:	SP=__________	PC=__________
Después:	SP=__________	PC=__________

Observamos que antes, SP = 01FD y PC = 010E, mientras que después, SP = = 01FF y PC = 0123. Los dos bytes situados en lo alto del stack han sido introducidos (POP) en el registro PC de forma que la ejecución puede continuar en la próxima instrucción, LD A,00H, después de la instrucción CALL DELAY.

Paso 5

Vamos ahora a analizar cuando se deben utilizar subrutinas. Se deben aplicar dos criterios al tomar una decisión concerniente a cuando un grupo de instrucciones debe ser o no una subrutina:

1. *Criterio funcional:* El conjunto de instrucciones ¿forman una unidad lógica con entradas y salidas bien definidas? Es decir, tiene sentido separar la función del grupo de instrucciones del resto del programa.
2. *Criterio de eficiencia:* ¿Cuesta más en términos de tiempo y espacio de memoria convertir un conjunto de instrucciones en una subrutina de lo que se merecen las consideraciones funcionales del primer criterio?
   Considere el siguiente "análisis en el peor de los casos":
   Supongamos que estamos analizando cuando se deben convertir o no un grupo de instrucciones equivalentes a M bytes de memoria en una subrutina y ser llamada entonces desde el programa del cual forma parte actualmente. Supongamos además que este grupo de instrucciones aparece R veces en el programa. Entonces,

   # de bytes utilizados como parte de un programa = (# veces que se produce en un programa) × (# bytes por vez)
   = R × M

Máximo # bytes utilizados como subrutina	= (# bytes en una subrutina incluyendo RET)
	+ (# llamadas en el programa) × (# de bytes por llamada)
	+ (3 bytes para colocar el stack)
	+ (dos bytes de stack para la dirección de retorno)
	= M + (# RETnos) + 3 R + 3 + 2
	= M + (# RETnos) + 3 R + 5

El criterio de eficiancia de espacio se satisface mientras:

$$R \times M > M + (\#RETs) + 3R + 5$$

Con respecto al criterio de eficiencia de tiempo, la utilización de subrutinas es siempre más lenta que la repetición de conjuntos de instrucciones muchas veces en un programa. La instrucción CALL es una de las que consume una mayor cantidad de tiempo en el conjunto de instrucciones del Z-80. La razón para esto estriba en que la CPU no solamente debe leer tres bytes de la memoria para interpretar la instrucción y cambiar el registro PC, sino que además el antiguo valor del PC debe ser guardado en el stack.

La instrucción de retorno consume tiempo porque debe extraer (POP) la información del stack para cambiar el registro PC. En cada grupo de instrucciones que se extraen del flujo secuencial de un programa para formar una subrutina, el tener que añadir las instrucciones CALL y RET constituye una sobrecarga en términos de tiempo. En muchos casos, el espacio ahorrado no compensa las consideraciones de tiempo. En los casos en que el tiempo es crítico se puede utilizar la instrucción RST, que es ligeramente más rápida, o en el extremo se puede sacrificar la eficiencia en el espacio en interés de la velocidad.

Paso 6

Analice el programa de muestra dado en este experimento para determinar si la utilización de una subrutina ha sido eficiente en términos de tiempo y espacio. Defienda su respuesta.

Nuestra respuesta es que, basándonos estrictamente en criterios de espacio, no fue una buena idea utilizar una subrutina. Puesto que la subrutina DELAY solamente se llama una vez, el espacio de memoria utilizado es menor para la implementación "sin-subrutina":

```
            LD B,03H
DELAY:      LD C,FFH
LOOP2:      LD C,FFH
LOOP3:      DEC D
            JR NZ,LOOP3
            DEC C
            JR NZ,LOOP2
            DJNZ DELAY
            LD A,00H
            RST 38H
```

total de bytes = 19

Ahorros = (# de bytes utilizados en el programa de muestra y subrutina)
+ 2 bytes de stack para la dirección de retorno −19 = 9 bytes

Nota: 9 = (2 bytes para el stack) + (3 bytes para LD SP,<B3>
+ (3 bytes para CALL DELAY) + (1 byte para RET).

NUNCA ES MAS RAPIDO utilizar una subrutina que insertar las instrucciones deseadas en línea, sin bifurcación. Sin embargo, un sacrificio en la velocidad o un sacrificio en la utilización de la memoria (como sería necesario en el ejemplo anterior) está a menudo compensado por las ventajas inherentes de escribir programas modulares bien estructurados que son más fáciles a depurar, a mantener y modificar.

Paso 7

Ejecutar el programa de llamada variando el valor del parámetro de temporización que se pasa a la subrutina mediante el registro B.

EXPERIMENTO N.° 3

Proposito

El propósito de este experimento es demostrar la instrucción RST N.

Programa N.° 22

Posición de memoria	Código objeto	Código fuente	Comentarios
0130	31 FF 01	LD SP,01FFH	; Situar el stack del sistema

0133	01 00 01	LD BC,0100H	; BC = # bytes de memoria a poner a 0
0136	21 00 04	LD HL,0400H	; HL = dirección de inicio
0139	D7	RST 16	; Llamar a la rutina de puesta a cero
013A	FF	RST 38H	; Devolver el control al sistema operativo

Subrutina

Posición de memoria	Código objeto	Código fuente	Comentarios
0010	F5	PUSH AF	; Guardar los registros como estaban
0011	C5	PUSH BC	; antes de la llamada a la subrutina
0012	D5	PUSH DE	
0013	E5	PUSH HL	
0014	36 00	LD (HL),00H	; Cargar la primera posición con ceros
0016	54	LD D,H	; Mover la dirección en el par de
			; registros HL a DE
0017	5D	LD E,L	
0018	13	INC DE	; Incrementar DE
0019	ED B0	LDIR	; Poner a cero todas las posiciones a partir de DE
001B	06 03	LD B,03H	; Colocar el byte de temporización para un retardo
001D	CD 02 01	CALL DELAY	; Llamar a la rutina DELAY
0020	F1	POP HL	; Restaurar los registros al estado original
0021	C1	POP DE	; antes de la llamada a la subrutina
0022	D1	POP BC	
0023	E1	POP AF	
0024	C9	RET	

Paso 1

Cargar el programa anterior y la subrutina en las direcciones indicadas. Asegurarse que la subrutina DELAY del experimento número 2 esté también cargada en la memoria.

Paso 2

El programa principal (listado bajo el título PROGRAMA N.° 22) utiliza los pares de registros BC y HL para pasar dos parámetros a la subrutina situada en 0010. BC contiene el número de bytes de memoria en los cuales se va a cargar 00. HL contiene la dirección del primer byte, y la subrutina utiliza la instrucción LDIR para poner a cero "BC" posiciones consecutivas de memoria empezando en la posición "HL". Nótese que debido a la dirección de inicio especial de la subrutina, 0010, hay dos instrucciones CALL que se utilizan para llamar a la subrutina, CALL 0010H y RST 16H. La instrucción CALL 0010 ocupa tres bytes de memoria puesto

que su código asociado es CD 10 00. La instrucción RST 16, cuyo código hex es D7, solamente un byte. Las dos realizan una función idéntica.

La subrutina situada en 0100 utiliza una técnica estándar para conservar los registros A, B, C, D, E, H y L exactamente como estaban cuando se llama a la subrutina. Las primeras cuatro instrucciones de la subrutina, PUSH (introducen) los registros en el stack. Las últimas cuatro instrucciones antes de RETornar restauran el estado de los registros utilizando cuatro instrucciones POP. Ponga una atención especial a la relación existente entre el orden en el cual los registros han sido introducidos (PUSH) y extraídos (POP) en el stack. El orden en el cual se efectúa el PUSH es exactamente el inverso del orden en el que se efectúa el POP. La razón para esto está en la disciplina de procesamiento del stack que responde a una pila LIFO el ûltimo-dentro-el-primero-fuera. Cuando el efecto producido por una subrutina en todos los registros es nulo, se dice que la subrutina *preserva el estado de la CPU.* Así, la subrutina en 0010 preserva el estado de la CPU, mientras que la subrutina DELAY (en la posición 0102) no lo hace.

Nótese que la subrutina en 0100 a su vez llama a otra subrutina, ver la instrucción CALL DELAY en la posición 001D. Esta se llama una *llamada a subrutina*

00 00		
00 10 00 20 00 25	SUBRUTINA	Carga BC bytes de memoria empezando en la posición HL (preserva el estado de la CPU)
00 30 01 00		
01 02 01 0D	SUBRUTINA DELAY	Retardo dependiente del contenido del registro B
01 10 01 20		
01 30 01 3D	PROGRAMA PRINCIPAL	Todas las rutinas son llamadas desde este programa
01 40		
01 E0 01 E8 01 F0 01 F8 01 FF 02 00	STACK	
04 00 05 00	DATO	El bloque de memoria que se va a cargar con ceros

Figura 8-6. Mapa de la memoria.

anidada. Así, no existe una regla en contra de que las subrutinas llamen a subrutinas. ¡Las subrutinas pueden llamarse hasta a sí mismas! Esta técnica de *programación recursiva,* en la cual las subrutinas se llaman a sí mismas, es muy potente, pero también muy difícil a entender este concepto. No lo discutiremos aquí ahora.

¿Qué sucede con las llamadas a subrutina anidadas? Básicamente las direcciones de retorno se van almacenando en el stack hasta que finalmente una subrutina efectúa un retorno RET, RET NZ, RET Z, RET C, RET NC, RET P, RET M, RET PE, o RET PO, en cuyo momento se extrae una dirección del stack (POP).

Ejecute el programa en la dirección 0130. Inserte puntos de paro en el programa en forma juiciosa de forma que usted pueda observar como crece el stack y como decrece a medida de que avanza la ejecución.

Paso 3

Normalmente, un programa, dos subrutinas, un stack, y un bloque de memoria para los datos del programa ocupan memoria de lectura/escritura. A menudo es una excelente idea guardar un mapa con la utilización de la memoria. La figura 8-6 da un mapa de la memoria para este experimento.

EXPERIMENTO N.° 4

Propósito

El propósito de este experimento es el de demostrar cuatro técnicas para pasar parámetros a las subrutinas.

Técnica N.° 1

Paso de parámetros mediante los registros.

Programa N.° 23

Posición de memoria	Código objeto	Código fuente	Comentarios
0200	31 FF 08	LD SP,08FFH	; Localizar el stack del sistema
0203	01 00 01	LD BC,0100H	; BC = bytes de memoria a poner a cero
0206	21 00 04	LD HL,0400H	; HL = dirección de inicio
0209	CD 10 02	CALL ZERO1	; Llamar la rutina de puesta a cero
020C	FF	RST 38H	; Devolver el control al sistema operativo

Subrutina

Posición de memoria	Código objeto	Código fuente		Comentarios
0210	36 00	ZERO1:	LD (HL),00H	; Colocar la primera posición a 00
0212	54		LD D,H	; Mover la dirección del par
0213	5D		LD E,L	; de registros HL a DE
0214	13		INC DE	; Incrementar DE
0215	ED B0		LDIR	; Poner a cero las posiciones empezando en DE
0217	C9		RET	; Devolver el control al programa principal

Técnica N.° 2

Paso de parámetros mediante el stack.

Programa N.° 24

0220	31 FF 08		LD SP,08FFH	
0223	01 00 01		LD BC,0100H	
0226	21 00 04		LD HL,0400H	
0229	C5		PUSH BC	; Entrar (PUSH) los parámetros en el stack
022A	E5		PUSH HL	
022B	CD 31 02		CALL ZERO2	
022E	FF		RST 38H	

Subrutina

0231	D1	ZERO2:	POP DE	; Pop la dirección de retorno del stack
0232	E1		POP HL	; Pop parámetros del stack
0233	C1		POP BC	
0234	D5		PUSH DE	; Push la dirección de retorno de nuevo
0235	36 00		LD (HL),00H	; en el stack

Técnica N.° 3

Paso de parámetros mediante un bloque de control en la memoria.

Programa N.° 25

0240	31 FF 08	LD SP,08FFH	
0243	01 00 01	LD BC,0100H	
0246	21 00 04	LD HL,0400H	
0249	ED 43 00 08	LD (0800H),BC	; Guardar los parámetros en un ; bloque de control de la memoria
024D	22 02 08	LD (0802H),HL	
0250	CD 56 02	CALL ZERO3	
0253	FF	RST 38H	

Subrutina

```
0256   ED 4B 00 08   ZERO3: LD BC,(0800H)   ; Cargar los parámetros guardados
025A   2A 02 08             LD HL,(0802H)   ; en la memoria a los registros
025D   36 00                LD (HL),00H     ; adecuados
025F   54                   LD D,H
0260   5D                   LD E,L
0261   13                   INC DE
0262   ED B0                LDIR
0264   C9                   RET
```

Técnica N.° 4

Paso de parámetros mediante posiciones de memoria situadas inmediatamente después de la instrucción CALL.

Programa N.° 26

```
0265   31 FF 08             LD SP,08FFH
0268   CD 72 02             CALL ZERO4
026B   00 01                DEFW 0100H      ; DEFW es un "pseudo operadór"
026D   00 04                DEFW 0400H      ; implementado por muchos
                                            ; ensambladores,
026F   FF                   RST 38H         ; para permitir que los datos se puedan
                                            ; mezclar con las instrucciones. Ver el
                                            ; próximo capítulo para una explicación
                                            ; completa.
```

Subrutina

```
0272   DD E1       ZERO4: POP IX            ; IX = dirección del primer parámetro
0274   DD 4E 00           LD C,(IX)
0277   DD 46 01           LD B,(IX+01H)     ; Cargar los datos en los registros
027A   DD 6E 02           LD L,(IX+02H)
027D   DD 66 03           LD H,(IX+03H)
0280   11 04 00           LD DE,0004H       ; Sumar 0004 a IX para obtener la dirección
0283   DD 19              ADD IX,DE         ; de retorno de la subrutina
0285   DD E5              PUSH IX           ; Entrar la dirección de retorno en el stack
0287   36 00              LD (HL),00H
0289   54                 LD D,H
028A   5D                 LD E,L
028B   13                 INC DE
028C   ED B0              LDIR
028E   C9                 RET
```

Paso 1

Cargar los cuatro conjuntos de programas y subrutinas. Verificar que se han cargado correctamente.

Paso 2

Estudiar las cuatro técnicas de pase de parámetros cuidadosamente asegurándose de entender como trabaja cada una de ellas. Obsérvese que cada pareja de programa-subrutina realiza exactamente la misma función. También, que no se puede utilizar cualquier programa para llamar a cualquier subrutina. Los programas y subrutinas están aparejados en el sentido de que debe tener efecto una coordinación en como se pasarán los parámetros entre el programa que llama a la subrutina y a la misma subrutina. ¿Cómo pueden compararse estas técnicas? Vamos primero a mirar las necesidades de la memoria.

Técnica	Programa	Subrutina	N.° total de bytes
1	15	8	23
2	17	12	29
3	22	15	37
4	13	27	40

Aunque la Técnica N.° 1 tiene las menores necesidades de memoria en este grupo de ejemplos, se podrían fácilmente realizar ejemplos en los cuales la Técnica N.° 4 fuera más corta. Cada vez que se llama la subrutina ZERO1, se ocupan nueve bytes de espacio en el programa que llama a la subrutina para colocar los parámetros y llamar a la subrutina. Para disponer los parámetros y llamar a la subrutina ZERO4 solamente se ocupan siete bytes. El mayor número de bytes añadidos a ZERO4 para implementar la técnica más complicada es *UN COSTE DE UNA SOLA VEZ.* Cuantas más veces se llama a ZERO4, son menos significativos los bytes adicionales de la subrutina, y son mayores los ahorros obtenidos en la secuencia de llamada.

Para utilizar la Técnica N.° 1, todos los parámetros deben caber dentro de los registros disponibles. Algunas veces esto puede ser una seria restricción. Las técnicas números 2 y 3 utilizan ambas memoria para pasar los parámetros. La Técnica N.° 2 es excelente para pasar grupos de parámetros que se utilizan y son descartados por la subrutina en una secuencia particular. La Técnica N.° 3 es la única forma práctica de pasar matrices y largas cadenas de caracteres entre las subrutinas.

Paso 3

Ejecute todos los pares de programa-subrutina. Ponga especial atención en la Técnica N.° 4 la cual es menos evidente que el resto.

EXPERIMENTO N.° 5

Propósito

El propósito de este experimento es el de demostrar la utilización de tablas de salto.

Programa

Posición de memoria	Código objeto	Código fuente	Comentarios
0900	61	DEFB 61H	; Valor # 1
0901	41 09	DEFW 0941H	; Dirección para Proceso # 1
0903	62	DEFB 62H	; Valor # 2
0904	45 09	DEFW 0945H	; Dirección para Proceso # 2
0906	63	DEFB 63H	; Valor # 3
0907	50 09	DEFW 0950H	; Dirección para Proceso # 3
0909	64	DEFB 64H	; Valor # 4
090A	55 09	DEFW 0955H	; Dirección para Proceso # 4
090C	00	DEFB 00H	; INDICADOR DE FINAL DE TABLA DE SALTO
0915	21 FD 08	START:LD HL,08FDH	; Inicializar HL
0918	23	NEXT: INC HL	; Incrementar HL para señalar al valor de entrada
0919	23	INC HL	; en la tabla de salto
091A	23	INC HL	
091B	7E	LD A,(HL)	; Cargar valor en el acumulador
091C	B7	OR A	; Colocar a 1 el indicador de cero si A es cero
091D	CC 38 00	CALL Z,0038H	; Si A = 0, entonces se ha alcanzado el final de ; la tabla de salto: pasar control al sistema ; operativo
0920	B8	CP B	; ¿Es B = A?
0921	20 F5	JR NZ,NEXT	; Si no, probar el próximo valor en la tabla ; de salto
0923	23	INC HL	; Si es así, las dos próximas posiciones contienen ; la dirección de la rutina a ejecutar
0924	5E	LD E,(HL)	; Cargar la dirección en DE–
0925	23	INC HL	; Primero LO, y después HL
0926	56	LD D,(HL)	
0927	62	LD H,D	; Mover el contenido de DE a HL
0928	6B	LD L,E	
0929	E9	JP (HL)	; Saltar a la dirección en HL

Paso 1

Vamos primero a discutir las nuevas instrucciones que aparecen en el programa anterior. Para entender DEFB y DEFW, examine cuidadosamente el código hex asociado con estas instrucciones. Por ejemplo DEFB 41H tiene 41 como su código hex asociado con estas instrucciones. En general,

DEFB <B1>

tiene <B1> como su código hex asociado. La instrucción DEFB significa DEFine Byte porque todo lo que hace la instrucción es dar un byte que es colocado directamente en la memoria. Una instrucción DEFW tiene un efecto similar para direcciones de dos bytes:

DEFW <B2><B1>

tiene <B1> <B2> como su código hex asociado. Las instrucciones DEFW significa DEFine Word, en donde *word* (palabra) en este contexto es una dirección absoluta de dos bytes. Es importante apreciar una diferencia muy sutil entre las instrucciones DEFB o DEFW y todas las otras instrucciones discutidas anteriormente. Todas las instrucciones que hemos discutido anteriormente corresponden a operaciones que realiza la CPU del Z-80. DEFB y DEFW *no* corresponden a operaciones realizadas por el Z-80, sino que provocan la inserción directa de datos en la memoria. De esta forma el ensamblador (tanto si es una persona como un sistema de software) inicializa la memoria a los valores deseados.

Las definiciones DEFB y DEFW que se acaban de describir son llamados *códigos pseudo-operadores del lenguaje ensamblador.* La razón por la cual estas instrucciones son llamadas códigos pseudo-operadores, es debido a que no son instrucciones ejecutables por el Z-80. Por el contrario, son instrucciones que ejecuta el ENSAMBLADOR para generar el código objeto. Otro pseudo-operador implementado en muchos ensambladores es la instrucción DEFINE STORAGE (definir almacenamiento) cuyo formato es

DEFS n

El pseudo-operador DEFS se utiliza para decirle al ensamblador que reserve un número especificado, n, de bytes en el código objeto para almacenamiento. El ensamblador no inserta valores particulares en el espacio de almacenamiento, sino que solamente salta a los próximos n bytes antes de continuar a cargar la memoria con el código objeto generado. El número n puede ser un número hexadecimal (seguido con una H) o un número decimal (seguido por un punto).

La instrucción CP B en la posición 0920 se discutirá en detalle en un capítulo siguiente. El código hex asociado para esta instrucción es B8. La instrucción compara el contenido del registro B con el del registro A. Si son iguales, el indicador de cero es colocado a 1 lógico, de lo contrario se coloca a cero lógico. Así el programa de este experimento utiliza las instrucciones CP B y JR NZ para determinar si son iguales A y B y bifurca en dos direcciones distintas de acuerdo con el resultado.

Paso 2

Una tabla de salto es un método muy eficiente para implementar una lógica de bifurcación en un programa que se parece mucho al ejemplo de la figura 8-7.

Esto es, un proceso termina con un cierto resultado, y entonces basándose en este resultado se realiza uno entre otros muchos procesos (N anterior, 4 en la tabla de salto de muestra). El término de software para esto es el *análisis de caso.* Cada resultado posible constituye un *caso;* si en CASO el resultado es X, se realiza el proceso J. La tabla de salto del ejemplo dado previamente tiene el formato general:

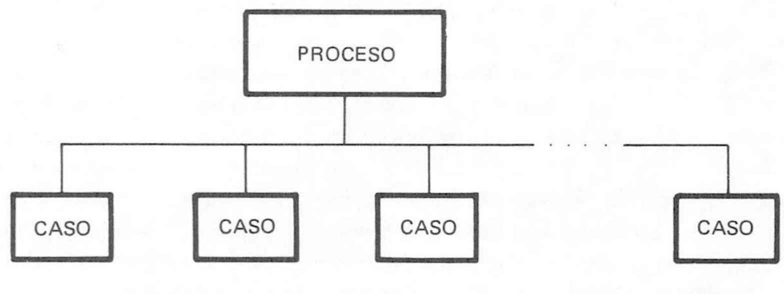


Figura 8-7. Tabla de salto.

VALOR 1
DIRECCION 1
VALOR 2
DIRECCION 2
VALOR 3
DIRECCION 3
•
•
•
VALOR N
DIRECCION N
00

Cada resultado posible es listado con la dirección de su rutina asociada siguiendo a continuación. 00 denota el final de la tabla de salto. Para realizar un análisis de caso, el resultado (en el registro B) es comparado (CP B) con cada *valor* de la tabla de salto hasta que se encuentra una igualdad o hasta que se alcanza el final de la tabla. Al encontrar una igualdad, el control se pasa a la dirección que sigue inmediatamente al byte VALOR saltando a la dirección contenida en el par de registros HL.

Almacene un salto al sistema operativo (FF) en cada dirección de la tabla de salto. Normalmente, se guardará un grupo de instrucciones con un propósito específico en estas direcciones.

Cargar el registro B (mediante el sistema operativo) con valores tales como 45, 55 y 41 los cuales están en la tabla de salto, y 01 o 09 que no están en la tabla de salto, y ejecutar el programa en modo paso a paso empezando en 0915. ¿Qué sucede si usted empieza la ejecución en 0900?

Entonces usted está ejecutando sus datos. Esta no es casi nunca una buena idea. Recuerde la diferencia fundamental entre los datos y el código ejecutable (el computador no puede hacerlo).

Paso 3

Existen métodos alternativos al que se ha presentado en el Paso 2 para implementar tablas de salto. Un método común y eficiente es el de segregar los valores y las direcciones en dos tablas separadas.

VALOR 1	DIRECCION 1
VALOR 2	DIRECCION 2
VALOR 3	DIRECCION 3
•	•
•	•
•	•
VALOR N	DIRECCION N

Primero se busca en la tabla de valores una igualdad. Si el valor N tiene una coincidencia entonces se pasa el control a la rutina que está en la dirección N en la tabla de direcciones. Trate de implementar este método de análisis de caso, como un ejercicio. Aquí hay algunas ayudas:

1. Utilice 00 para marcar el final de cada tabla. Nótese que si 00 es un valor posible o byte de dirección, se debe elegir un nuevo indicador de final, o se debe diseñar un nuevo método para detectar el final de la tabla.
2. Observe que la tabla de valores tiene los items de un byte mientras que la tabla de direcciones los tiene de dos bytes.
3. Como en el programa de muestra, suponga que el valor a comparar ya está en uno de los registros de la CPU (elija el más conveniente).
4. Utilice las instrucciones DEFB y DEFW para confeccionar las tablas.

Una vez que haya escrito el programa, cárgelo y ejecútelo. Entonces verá que tal lo ha hecho.

9

Instrucciones lógicas

La tabla 9-1 contiene las Instrucciones Aritméticas y Lógicas. La tabla 9-2 contiene las instrucciones del Z-80 que manipulan solamente el acumulador y los indicadores, las instrucciones AF de uso general. Explicaremos estas instrucciones contenidas en estas tablas en este capítulo y en el capítulo 11. Este capítulo le introducirá a las instrucciones lógicas que realizan operaciones lógicas en palabras binarias de 8 bits. Los mnemónicos para estas instrucciones son AND, XOR, OR y CPL. Debido a que se necesita una comprensión completa de las operaciones lógicas de varios bits, para utilizar con eficacia estas instrucciones, hemos incluido un material introductorio para este tema.

Al completar este capítulo, usted será capaz de hacer lo siguiente:

- Resumir las tablas de verdad para las operaciones lógicas de un bit, AND, OR, XOR y NOT.
- Dibujar los símbolos lógicos correctos del álgebra de Boole para AND, OR, XOR y NOT.
- Explicar cómo se realizan operaciones lógicas multibit.
- Realizar las operaciones lógicas AND, OR y XOR en pares de bytes de datos de 8 bits.
- Escribir el teorema de De Morgan en el álgebra de Boole.
- Establecer el teorema de De Morgan utilizando símbolos lógicos.
- Listar las instrucciones lógicas en el conjunto de instrucciones del Z-80.

- Explicar cómo se pueden utilizar las instrucciones lógicas en un programa de microcomputador.
- Definir una *máscara.*

Tabla 9-1. Grupos aritmético y lógico de 8 bits

	ORIGEN										
	DIRECCIONAMIENTO POR REGISTRO							REG. INDIR.	INDEXADO		INMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
SUMAR CON ARRASTRE 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
RESTAR 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
RESTAR CON ARRASTRE 'SUB'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARAR 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENTAR 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENTAR 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Cortesía Zilog, Inc.

¿QUE ES UNA INSTRUCCION LOGICA?

En esta sección trataremos de las operaciones lógicas AND, OR, XOR (O exclusiva) y CPL (complemento). Se realiza una operación lógica o instrucción lógica con dos bytes de datos de 8 bits, sujetándose los bits correspondientes de cada byte a una operación lógica de 2 bits tal como AND, OR o XOR. En el microprocesador Z-80 un byte de datos está en el acumulador y el resultado final se guarda en el acumulador. Esta es una explicación de porque llamamos al *acumulador* un "ACU-

Tabla 9-2. Operaciones AF de utilización general

Ajuste decimal del Ac. 'DAA'	27
Complementar Ac, 'CPL'	2F
Compl. Ac. 'NEG' (complemento a dos)	ED 44
Complementar al indicador de arrastre, 'CCF'	3F
Poner a 1 el indicador de arrastre, 'SCF'	37

Cortesía Zilog, Inc.

MULADOR"; *acumula* el resultado final de las operaciones aritméticas o lógicas. La instrucción lógica de un byte, CPL, opera directamente en el acumulador y no incluye ningún otro registro o célula de memoria.

Tabla de verdad para las operaciones lógicas de un bit

AND

B	A	Q
0	0	0
0	1	0
1	0	0
1	1	1

OR

B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

XOR

B	A	Q
0	0	0
0	1	1
1	0	1
1	1	0

NOT

A	Q
0	1
1	0

Llamamos a estas tablas de verdad "tablas de 1 bit" debido a que las palabras de datos A y B, contienen cada uno un solo bit. XOR es una abreviación de Exclusive-OR (O Exclusivo).

ALGEBRA DE BOOLE

Cuando discutimos instrucciones lógicas, es útil emplear *símbolos booleanos.* Estos símbolos proceden del tema del *álgebra de Boole,* que es la matemática de los sistemas lógicos. Se utilizan símbolos alfabéticos tales como A, B, C, . . ., Q para representar las variables lógicas y los símbolos 1 y 0 se utilizan para representar los estados lógicos. Esta forma particular de las matemáticas fue creada en Inglaterra por George Boole en 1847. No fue utilizada extensivamente hasta 1938, cuando Claude Shannon la adaptó para analizar redes de multicontactos para sistemas telefónicos.

Lo que usted debe aprender acerca del álgebra de Boole son los símbolos lógicos básicos que son utilizados en los cálculos en el álgebra de Boole y que también son

utilizados en la lógica digital. Estos símbolos incluyen los siguientes:

- \+ el cual significa la suma lógica y recibe el nombre OR (O)
- ● que significa la multiplicación lógica y que recibe el nombre de AND (Y)
- ⊕ que recibe el nombre O-Exclusiva o XOR
- – que significa el complemento lógico y que recibe el nombre NOT (NO).

El símbolo de complemento es una barra continua encima de la variable lógica tal como A, B, C, . . ., Q. Así la definición en álgebra de Boole para un estado AND de dos entradas es Q = A ● B, o simplemente Q = AB. En la figura 9-1 se dan las definiciones para diferentes tipos de puertas. Obsérvese la utilización de la barra –, para las puertas NAND y NOR.

Es útil resumir el símbolo de las operaciones para las cuatro operaciones lógicas que estamos considerando:

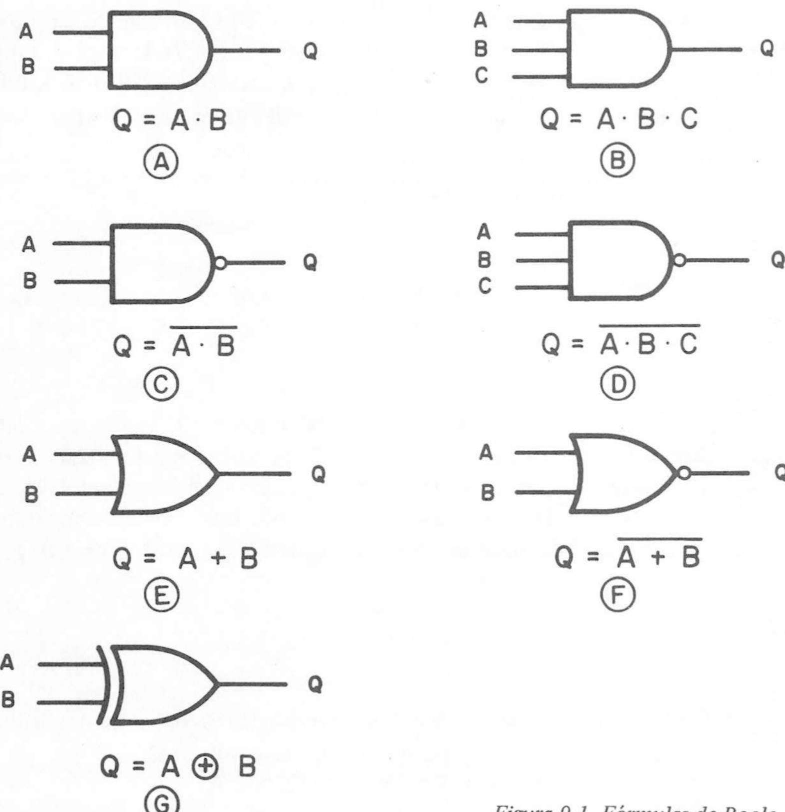


Figura 9-1. Fórmulas de Boole.

AND	OR	XOR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$0 \oplus 0 = 0$	$\overline{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$0 \oplus 1 = 1$	$\overline{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	$1 \oplus 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	$1 \oplus 1 = 0$	

Estas son operaciones lógicas de 1 bit.

OPERACIONES MULTIBIT

Las operaciones lógicas multibit son tratadas como muchas operaciones lógicas de un bit. No intervienen nuevos principios de lógica. Los bits correspondientes de una palabra binaria operan lógicamente con los bits correspondientes de la segunda palabra binaria para producir un resultado multibit conjunto. Considere la variable lógica de ocho bits A. Los bits individuales en la palabra de 8 bits se pueden etiquetar como A7, A6, A5, A4, A3, A2, A1 y A0, siendo A0 el bit menos significativo (el bit 2**0) y siendo A7 el bit más significativo (el bit 2**7). Considere también la variable lógica de 8 bits, B, la cual tiene los bits individuales B7, B6, B5, B4, B3, B2, B1 y B0, siendo B0 el bit menos significativo y B7 el más significativo. La operación lógica A ● B = Q, significa lo siguiente:

$$
\begin{aligned}
A0 \cdot B0 &= Q0 \\
A1 \cdot B1 &= Q1 \\
A2 \cdot B2 &= Q2 \\
A3 \cdot B3 &= Q3 \\
A4 \cdot B4 &= Q4 \\
A5 \cdot B5 &= Q5 \\
A6 \cdot B6 &= Q6 \\
A7 \cdot B7 &= Q7
\end{aligned}
$$

El resultado de la operación lógica es la variable lógica Q, que tiene como bit menos significativo Q0 y el bit más significativo Q7. En otras palabras, *las operaciones lógicas multibit se realizan bit a bit en una serie de operaciones lógicas de 1 bit.*

Es más fácil realizar operaciones lógicas de muchos bits si las palabras binarias multibit se colocan una debajo de la otra. Así, si A = 11110000 y B = 00111100, entonces AB es

$$
\begin{array}{r}
11110000 \\
\underline{00111100} \\
00110000
\end{array}
$$

o Q = 00110000. Hemos realizado una operación lógica AND, y hemos utilizado las relaciones 0 ● 1 = 0 y 1 ● 1 = 1 para obtener el resultado final.

De una forma similar, la operación lógica multibit, A + B = Q, significa lo siguiente:

$$A0 + B0 = Q0$$
$$A1 + B1 = Q1$$
$$A2 + B2 = Q2$$
$$A3 + B3 = Q3$$
$$A4 + B4 = Q4$$
$$A5 + B5 = Q5$$
$$A6 + B6 = Q6$$
$$A7 + B7 = Q7$$

De nuevo, el resultado de la operación lógica es la variable lógica Q, que contiene ocho bits. Si A = 11110000 y B = 00111100, entonces Q = A + B valdrá

$$\begin{array}{r} 11110000 \\ \underline{00111100} \\ 11111100 \end{array}$$

o Q = 11111100. Hemos realizado una operación lógica OR, y hemos utilizado las relaciones 0 + 1 = 1, 1 + 1 = 1 y 0 + 0 = 0 para obtener el resultado final. *Nótese que el símbolo + representa la operación lógica* OR *y la operación aritmética "más".* Aquí hay una posibilidad de confusión y usted debe tener cuidado con ello.

La última operación lógica de interés es, A ⊕ B = Q que significa lo siguiente:

$$A0 \oplus B0 = Q0$$
$$A1 \oplus B1 = Q1$$
$$A2 \oplus B2 = Q2$$
$$A3 \oplus B3 = Q3$$
$$A4 \oplus B4 = Q4$$
$$A5 \oplus B5 = Q5$$
$$A6 \oplus B6 = Q6$$
$$A7 \oplus B7 = Q7$$

El resultado de esta operación O-Exclusiva es una variable lógica de 8 bits, Q. Si A = 11110000 y B = 00111100, entonces Q = A ⊕ B se convierte en

$$\begin{array}{r} 11110000 \\ \underline{00111100} \\ 11001100 \end{array}$$

o Q = 11001100. Hemos realizado la operación lógica O-Exclusiva y hemos utilizado las relaciones 0 ⊕ 0 = 0, 0 ⊕ 1 = 1, 1 ⊕ 0 = 1, y 1 ⊕ 1 = 0 para obtener el resultado final.

NOT (NO)

La operación lógica NOT complementa cualquier dígito binario o grupo de dígitos binarios. Si A = 11110000, entonces Q = 00001111.

TEOREMA DE DE MORGAN

Un importante teorema en el álgebra de Boole es el *teorema de De Morgan,* el cual puede ser escrito de dos formas diferentes:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Una forma más interesante del teorema de De Morgan se obtiene utilizando los símbolos lógicos (figura 9-2). Este es un resultado importante y que usted encontrará muy útil en la electrónica digital y en el interface de los microcomputadores.

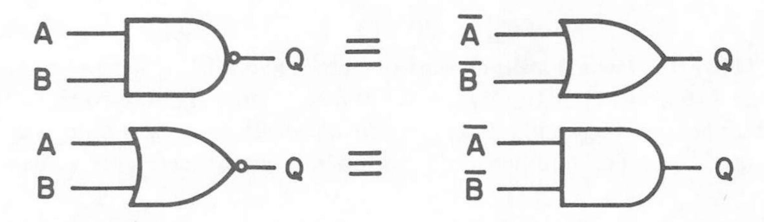


Figura 9-2. Símbolos lógicos.

Expresa que usted puede obtener una función lógica NOR (NI) negando todas sus entradas y aplicándolas a una puerta AND (Y); alternativamente usted puede realizar una función lógica NAND negando todas las entradas y aplicándolas a una puerta OR.

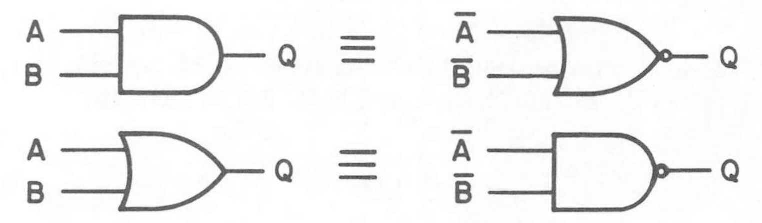


Figura 9-3. Símbolos lógicos.

El teorema de De Morgan también se puede representar mediante los símbolos lógicos de la figura 9-3. Estas expresan que usted puede realizar una función AND negando todas las entradas y aplicándolas a una puerta NOR; alternativamente, usted puede realizar una función OR negando todas las entradas y aplicándolas a una puerta NAND, los circuitos integrados con puertas NAND sòn muy baratos y usuales. El teorema de De Morgan demuestra como usted puede rápidamente crear puertas OR y NOR a partir de puertas NAND. Discutiremos las puertas NAND, NOR, AND, OR y otras puertas en detalle en el Libro 2.

GRUPO DE INSTRUCCIONES LOGICAS DEL Z-80

Todas las operaciones lógicas que hemos discutido son implementadas por la CPU del Z-80 como instrucciones pertenecientes al grupo de instrucciones lógicas. Vamos a investigar estas instrucciones en detalle. Ponga una atención particular a la manera en que estas instrucciones afectan al registro F de indicadores de estado puesto que este aspecto de su funcionamiento es a menudo esencial a su uso efectivo.

COMPLEMENTAR EL ACUMULADOR: CPL

Complementar el acumulador consiste en realizar una operación NOT (NO) en el byte de 8 bits del acumulador. Esta instrucción de un solo byte, tiene el código hex de 2F y un código mnemónico de CPL. Los indicadores de arrastre cero, P/V, y signo no quedan afectados por esta instrucción, por ejemplo:

	Contenido del acumulador
Antes de la ejecución de CPL	10111010
Después de la ejecución de CPL	01000101

Una buena aplicación de la instrucción CPL es la siguiente:

```
CPL A
INC A
```

Estas dos instrucciones encuentran el *complemento a dos* de ocho bits del acumulador.

AND CON EL ACUMULADOR: AND

Las 11 instrucciones AND distintas de que dispone el conjunto de instrucciones del Z-80 tienen el mnemónico general AND S en donde S depende del modo de direccionamiento. El indicador de arrastre es puesto a cero por esta instrucción y los indicadores de signo y de cero vienen afectados por el resultado de la operación. El indicador de P/V detecta la paridad del resultado y es colocado a 1 si la paridad (número de bits a 1) es par y es colocado a cero si es al revés. Por ejemplo, considere la ejecución de la instrucción AND B. Esta instrucción hace que el Z-80 realice la operación lógica AND entre el contenido del acumulador y del registro B. El resultado es almacenado en el acumulador. Por ejemplo:

	Acumulador	Registro B	Ind. S	Ind. Z	Ind. P/V
Antes de la ejecución de AND B	11001100	10001011	X	X	X
Después de la ejecución de AND B	10001000	10001011	1	0	1
(X = No importa)					

Obsérvese que mientras que la instrucción AND A, que realiza la operación lógica del acumulador consigo mismo, puede parecer que sea una instrucción inútil, tiene actualmente alguna utilidad. Los indicadores de cero y el indicador de signo quedan afectados por el resultado de esta instrucción. La instrucción AND A coloca a 1 el indicador de cero si y sólo si el contenido del acumulador es 00. Similarmente AND A, coloca el indicador de signo si, y solamente si el contenido del acumulador es un número negativo en complemento a dos. Por ejemplo:

		Indicadores		
	Acumulador	S	Z	P/V
Antes de la ejecución de AND A	10000001	X	X	X
Después de la ejecución de AND A	10000001	1	0	1
(X = No importa)				

Otra aplicación útil de la instrucción AND es la de implementar una técnica llamada máscara. El término *máscara* está definido de la siguiente forma:

enmascarar: Una técnica lógica en la cual ciertos bits de una palabra multibit se ponen a cero o se inhiben.

Una careta o máscara cubre parte de la cara. En el mismo sentido, una máscara, utilizada en una operación del computador, cubre todos o la mayor parte de los bits de una palabra multibit, dejando solamente los bits que son importantes para que continúe la ejecución del programa. Considere la siguiente secuencia de instrucciones:

```
LD A,(0F32H)
AND 01H
```

El indicador de cero se coloca a 1 o a 0 dependiendo del valor del bit menos significativo del byte que está en la posición de memoria 0F32. Esta secuencia de instrucciones utiliza una máscara para hacer un test del bit D0 de un byte en la memoria, siendo la máscara el byte 01 de la instrucción AND. Similarmente, esta técnica se puede utilizar para hacer un test de otros bits. Las máscaras se pueden utilizar para mirar a subconjuntos de bits en un byte. Por ejemplo la secuencia de instrucciones

```
LD A,(0F32H)
AND 0FH
```

pone a cero los cuatro bits más significativos y deja igual los cuatro bits menos significativos del acumulador, permitiendo que un programa mire solamente a los cuatro bits menos significativos del byte situado en 0F32.

O-EXCLUSIVA CON EL ACUMULADOR: XOR

El mnemónico general para la instrucción O-exclusiva es XOR S en donde S depende del modo de direccionamiento. Como con la instrucción AND el indicador de arrastre es colocado a cero, y los indicadores de signo, cero, y paridad/sobrepasamiento quedan afectados como resultado de la operación.

Por ejemplo, considere la instrucción XOR (HL) en donde el par de registros HL contiene la dirección 1AB6. Esta instrucción provoca que el Z-80 realice la operación lógica O-exclusiva entre el contenido del acumulador y el contenido de la posición de memoria señalada por el par de registros HL y deja el resultado en el acumulador.

				Indicadores		
	Acumulador	HL	(HL)	S	Z	P/V
Antes de la ejecución	10010001	1AB6	00110000	X	X	X
Después de la ejecución	10100001	1AB6	00110000	1	0	0

Así ejecutando la operación XOR del byte 00 con el contenido del acumulador deja al acumulador intacto, pero afecta los indicadores de signo y de cero dando una información. La instrucción XOR A pone a cero el acumulador mediante una instrucción de un byte y es así, preferible a LD A,00H. El proceso de hacer la O-exclusiva del acumulador con FF tiene el mismo efecto que la instrucción CPL excepto que los indicadores de arrastre, cero y signo quedan afectados.

OR CON EL ACUMULADOR: OR

Las once instrucciones OR tienen el mnemónico general OR S, en donde S depende del modo de direccionamiento. Los indicadores de estado están afectados por la instrucción OR en la misma manera que lo están para las instrucciones XOR y AND. Esto es, el indicador de arrastre es colocado a cero y los indicadores de signo, cero y paridad/sobrepasamiento quedan afectados de acuerdo con el resultado de la operación. Por ejemplo considere la instrucción OR (IX + 02H). Esta instrucción hace que el Z-80 realice una operación lógica OR entre el contenido del acumulador y el contenido de la posición de memoria situada dos bytes más adelante de la posición direccionada por el registro índice IX.

	Acumulador (binario)	IX (hex)	(IX + 02) (binario)	S	Indicadores Z	P/V
Antes de la ejecución	00000011	1AB4	00110010	X	X	X
Después de la ejecución	00110011	1AB4	00110010	0	0	1

Así, la instrucción OR no puede ser utilizada para poner a cero el acumulador. Como usted ya habrá visto en varios experimentos, se puede utilizar OR A para

determinar si el acumulador es cero. Una utilización excelente de la instrucción OR es para determinar si un par de registros es cero. Por ejemplo, la siguiente secuencia de instrucciones determina si DE = 0000:

```
LD A,D
OR E
```

La única forma en que la operación OR E puede poner a 1 el indicador de cero es si D y E son ambos 00. Usted vió aplicar esta técnica al par de registros BC en el último experimento del capítulo 6. La razón para que esta técnica sea tan útil es que las instrucciones de 16 bits, incrementar y decrementar *no afectan* a ninguno de los indicadores. Así, si el par de registros se utiliza como un contador de bucle, el indicador de cero se colocará a 1, si el par de registros equivale a 0000 que puede ser la condición de que se ha acabado el bucle.

INSTRUCCIONES LOGICAS Y CONTROL DE DISPOSITIVOS EXTERNOS

Las instrucciones lógicas le permiten determinar cuando los dispositivos externos están en marcha o parados o si se han producido unos determinados acontecimientos. Como un ejemplo, supongamos que usted utiliza los estados lógicos 0 y 1 para representar una de las siguientes situaciones:

A. Estado del dispositivo: conectado/desconectado
   Cero lógico = el dispositivo está desconectado
   Uno lógico = el dispositivo está conectado.
B. Ocurrencia de un acontecimiento
   Cero lógico = el acontecimiento no se ha producido
   Uno lógico = el acontecimiento se ha producido.

En un capítulo posterior, usted aprenderá a utilizar la instrucción IN del microcomputador para entrar ocho bits de dato en el acumulador. Usted aprenderá que puede utilizar cada uno de los bits para representar el estado conectado/desconectado de un dispositivo específico, o si se ha producido o no un acontecimiento específico. Con ocho bits, usted puede representar el estado de ocho dispositivos o acontecimientos distintos. Considere los ocho dispositivos siguientes, cada uno de los cuales puede estar conectado o desconectado indistintamente:

Bit 0: Dispositivo de medida de presión
Bit 1: Dispositivo de medida de temperatura

Bit 2: Dispositivo de medida de velocidad
Bit 3: Dispositivo de medida de flujo
Bit 4: Dispositivo de medida de tensión
Bit 5: Dispositivo de medida de corriente
Bit 6: Dispositivo terminal de entrada en ASCII (teclado)
Bit 7: Dispositivo terminal de salida ASCII (impresora o pantalla)

Estos ocho dispositivos tienen ocho bits asociados, los cuales pueden ser introducidos en el mismo instante de tiempo en el acumulador del chip microprocesador Z-80. Una vez dentro del microprocesador, usted puede utilizar las instrucciones lógicas para determinar cuando algunos dispositivos específicos están conectados o desconectados.

INTRODUCCION A LOS EXPERIMENTOS

Los siguientes experimentos están diseñados para demostrar lo que usted ha aprendido en el capítulo 9 acerca de las instrucciones lógicas. Los experimentos que usted realizará se pueden resumir de la siguiente forma:

Experimento N.°	Comentarios
1	Demostrar una rutina AND de 16 bits.
2	Demuestra la utilización de las instrucciones AND, CPL y XOR en aplicaciones de control de dispositivos. Estas instrucciones se utilizan para determinar cuando ha cambiado de estado un dispositivo, y si es así, en que sentido, de conectado a desconectado o de desconectado a conectado.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar un programa que realiza una operación lógica AND entre el contenido del par de registros BC y DE y deja el resultado en HL.

Programa N.° 28

Posición de memoria	Código objeto	Código fuente		Comentarios
0200	78	AND16:	LD A,B	
0201	A2		AND D	; AND de los ocho bits más significativos
0202	67		LD H,A	; Cargar el resultado en el registro H
0203	79		LD A,C	
0204	A3		AND E	; AND de los ocho bits menos significativos
0205	6F		LD L,A	; Cargar el resultado en el registro L
0206	B4		OR H	; Colocar a 1 el indicador de cero si el
				; contenido de HL es 00, de lo contrario
0207	FF		RST 38H	; colocar a cero el indicador de cero

Paso 1

Cargar el programa anterior en las direcciones indicadas. Inicializar el par de registros BC y DE a cada uno de los valores siguientes y ejecutar entonces el programa. Escriba sus observaciones para el contenido del par de registros HL y para los indicadores de cero, signo, P/V y arrastre.

Téngase en cuenta que el indicador de cero es el bit D6 del registro F, el indicador de signo es el bit D7, el indicador P/V es el bit D2 y el indicador de arrastre es el bit D0, como se ilustra a continuación:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

BC=0 0 1 1 0 0 1 1 1 1 0 0 1 0 0 1 o 33 C9
DE=1 1 1 1 0 0 0 0 0 1 1 1 0 0 1 1 o F0 73
HL=
Indicador de cero = Signo = P/V = Arrastre =

Observamos que HL = 30 41 y que los indicadores se colocaron de la siguiente forma:

Indicador de cero = 0 Signo = 0 P/V = 1 Arrastre = 0
BC=1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 o A5 A5
DE=1 1 0 0 1 1 0 0 0 1 0 1 1 1 1 1 o CC 5F
HL=
Indicador de cero = Signo = P/V = Arrastre =

Observamos que HL = 84 05 y que los indicadores se colocaron de la siguiente forma:

Indicador de cero = 0 Signo = 1 P/V = 0 Arrastre = 0
BC=1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 o FF 00
DE=0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 o 00 FF
HL=
Indicador de cero = Signo = P/V = Arrastre =

Observamos que HL = 00 00 y que los indicadores se colocaron de la siguiente forma

Indicador de cero = 1 Signo = 0 P/V = 1 Arrastre = 0

Paso 2

Escribir, cargar y depurar un programa para realizar una operación OR de 16 bits entre los pares de registros BC y DE, dejando el resultado en HL y colocando a 1 o a 0 el indicador de cero de acuerdo con la operación. Compruebe el programa para verificar si está correcto cargando datos de muestra en los registros BC y DE y comprobando lo que se ha previsto con respecto a los valores generados por el programa para HL y el indicador de cero.

EXPERIMENTO N.° 2

Propósito

El propósito de este experimento es el de determinar que dispositivos han cambiado de estado entre ocho posibles, entre dos lecturas de estado y en que dirección se ha producido el cambio de estado, conectado a desconectado o viceversa. Los ocho dispositivos se listan a continuación con sus números de bit asociados:

Bit 0: Dispositivo de medida de presión
Bit 1: Dispositivo de medida de temperatura
Bit 2: Dispositivo de medida de velocidad
Bit 3: Dispositivo de medida de flujo
Bit 4: Dispositivo de medida de tensión
Bit 5: Dispositivo de medida de corriente
Bit 6: Dispositivo de medida de nivel de líquido
Bit 7: Dispositivo de medida de frecuencia

Un 1 lógico indica que el dispositivo está conectado, y un 0 lógico indicará que el dispositivo está desconectado. La entrada de dos bytes de estado distintos es simulada mediante dos instrucciones LD r, <B2>.

Programa N.° 29

Posición de memoria	Código objeto	Código fuente	Comentarios
0100	06 88	LD B,88H	; Simular la entrada del byte de estado previo en el ; registro B
0102	3E 09	LD A,09H	; Simular la entrada del byte de estado actual en el ; acumulador
0104	4F	LD C,A	; Copiar el estado actual en el registro C
0105	A8	XOR B	; O-Exlusiva entre el contenido de A y B, en el resulta- ; do un 1 lógico indica los dispositivos cuyo estado ha ; cambiado
0106	57	LD D,A	; Guardar la información en el registro D
0107	A0	AND B	; AND entre el contenido de A y B, en el resultado, el ; 1 lógico indica cuales son los dispositivos que han ; cambiado de estado desde 1 a 0
0108	67	LD H,A	; Guardar esta información en H
0109	2F	CPL	; Complementar el acumulador, en el resultado el 1 ; lógico indica los dispositivos que NO han cambiado ; de estado de 1 a 0, es decir, los dispositivos que han ; mantenido su estado constante o cambiado desde 0 ; a 1
010A	A2	AND D	; AND entre el contenido de A y D. En el resultado ; el 1 lógico indica los dispositivos cuyo estado ha cam- ; biado desde 0 a 1
010B	6F	LD L,A	; Guardar esta información en el registro L
010C	FF	RST 38H	; Devolver el control al sistema operativo

Paso 1

Cargar el programa precedente en las posiciones de memoria indicadas. Verifique que usted ha cargado el programa correctamente.

Paso 2

Examinar el programa cuidadosamente para verificar el siguiente resumen:

a. Si el bit n en el registro D está a 1 lógico, entonces el dispositivo asociado ha cambiado de estado.
b. Si el bit n en el registro L está a 1 lógico, entonces el dispositivo asociado ha pasado desde 0 a 1.
c. Si el bit n en el registro H está a lógico 1, entonces el dispositivo asociado ha pasado de 1 a 0.
d. Si el bit n en el registro L está a 0 lógico, entonces el dispositivo asociado no ha cambiado de estado o ha pasado de 0 a 1.

e. Si el bit n en el registro H está a 0 lógico, entonces el dispositivo asociado o bien no ha cambiado de estado o ha pasado de 0 a 1.

Para resumir, con la ayuda de las instrucciones lógicas, usted puede contestar a los siguientes tipos de preguntas mediante programas con el microcomputador:

- El estado lógico del bit de estado ¿es 0 o 1?
- Cuando se le compara con el estado lógico previo, el bit de estado ¿ha cambiado o bien permanece igual?
- Si el estado lógico del bit de estado ha cambiado, el cambio ¿ha sido desde 0 a 1 o desde 1 a 0?

Paso 3

Antes de que usted ejecute el programa, realice las siguientes operaciones lógicas:

Supongamos 10001000 = Byte de estado previo
00001001 = Byte de estado actual

a. 1000010000 XOR 00001001

10001000 = Byte de estado previo
<u>00001001</u> = Byte de estado actual

El resultado de esta operación lógica le indica cuál de los dispositivos ha cambiado de estado. El dispositivo n ha cambiado de estado si y solamente si el bit n está a 1 en el resultado de la operación XOR.

b. 10000001 AND 10001000

10001000 = Byte de estado previo
<u>10000001</u> = Resultado del apartado (a) diciendo que dispositivo ha cambiado de estado.

El resultado de esta operación lógica le indica cuál de los dispositivos ha cambiado de estado desde 1 a 0. El dispositivo n ha cambiado de estado desde 1 a 0 si y solamente si n está a 1.

c. CPL 10000000, el complemento de 10000000

10000000 es el resultado del apartado (b) y le indica que dispositivos han cambiado de estado desde 1 a 0. Así el resultado de esta operación lógica le dice a usted los dispositivos que no han cambiado de estado de 1 a 0.

d. 01111111 AND 10000001
01111111 = resultado de (c)
10000001 = resultado de (a)

El resultado de esta operación lógica le dice cuales son los dispositivos que han cambiado de estado desde 0 a 1.

Paso 4

Ejecute el programa precedente en su microcomputador. ¿Qué información aparece en el registro L después de su ejecución?

Aquellos dispositivos que han cambiado de estado desde 0 a 1 tienen sus bits asociados a 1 en el registro L.

Paso 5

¿Qué información aparece en el registro H?

Aquellos dispositivos que han cambiado de estado desde 1 a 0 tienen sus bits asociados a 1 en el registro H.

Paso 6

¿Qué información aparece en el registro C?

El byte de estado actual, es decir, los dispositivos que están a 1 y los dispositivos que están a 0.

REPASO

Las siguientes preguntas le ayudarán a revisar el uso de las instrucciones lógicas, el álgebra de Boole y las operaciones multibit.

1. Realice las operaciones lógicas indicadas en álgebra de Boole con números multibit.

   a. 11001011 • 01011010
   b. 00100000 + 11011111
   c. 00100000 • 11011111
   d. 10101010 ⊕ 10100100
   e. CC • 0B
   f. A6 ⊕ 80
   g. 37 ⊕ 04
   h. 49 • 1B

2. Un byte de estado de ocho bits está asociado con ocho dispositivos diferentes:

   Bit 0: Dispositivo de medida de presión
   Bit 1: Dispositivo de medida de temperatura
   Bit 2: Dispositivo de medida de velocidad
   Bit 3: Dispositivo de medida de flujo
   Bit 4: Dispositivo de medida de tensión
   Bit 5: Dispositivo de medida de corriente
   Bit 6: Dispositivo de medida de nivel de líquido
   Bit 7: Dispositivo de medida de frecuencia

   Para los bytes de estado hex dados a continuación, indicar cual de los dispositivos dados en la pregunta N.° 2 están conectados. Un 1 lógico para el bit indicado significa que el dispositivo está conectado.

   a. 53
   b. 40
   c. 64
   d. 20
   e. 02
   f. 30
   g. 30
   h. C0
   i. 01
   j. 28

3. Para los bytes de estado hex dados a continuación, incluyendo el byte de estado previo y el byte de estado actual, utilice las técnicas del álgebra de Boole para determinar cuales de los ocho distintos dispositivos incluidos en la pregunta N.° 1 han cambiado de estado de 1 a 0 o de 0 a 1. Muestre sus cálculos en álgebra de Boole.

	Byte de estado previo	Byte de estado actual
a.	84	46
b.	27	63
c.	02	07
d.	A7	DB

Respuestas

1. a. 01001010
   b. 11111111
   c. 00000000

d. 00001110
e. 00001000 = 08
f. 00100110 = 26
g. 00110011 = 33
h. 00001001 = 09

2. a. dispositivo de medida de nivel de líquido, tensión, temperatura y presión
   b. dispositivo de medida de nivel de líquido
   c. dispositivo de medida de nivel de líquido, corriente y velocidad
   d. dispositivo de medida de corriente
   e. dispositivo de medida de temperatura
   f. dispositivo de medida de corriente y tensión
   g. dispositivos de medida de velocidad y temperatura
   h. dispositivos de medida de frecuencia y nivel de líquido
   i. dispositivo de medida de presión
   j. dispositivo de medida de corriente y caudal

3. a. Primero usted convierte los dos bytes en código binario.

84 = 10000100 (byte de estado previo)
46 = 01000110 (byte de estado actual)

A continuación, usted realiza una operación O-Exclusiva en estos dos bytes de datos.

$$10000100 \oplus 01000110 = 11000010$$

Utilizando el resultado 11000010, usted realiza una operación AND entre este y el byte de estado previo,

$$10000100 \bullet 11000010 = 10000000$$

Usted realiza una operación NOT en este resultado,

$$\overline{10000000} = 01111111$$

Finalmente, usted emplea el resultado de la operación NOT y realiza una operación AND entre él y el resultado de la operación inicial O-Exclusiva,

$$01111111 \bullet 11000010 = 01000010$$

Ahora podemos hacer las conclusiones apropiadas.
1. Los dispositivos de medida de frecuencia, nivel de líquido y temperatura han cambiado de estado.
2. El dispositivo de medida de frecuencia pasó desde 1 a 0.
3. Los dispositivos de medida de nivel de líquido y de temperatura pasaron desde 0 a 1.

Inspeccionando los dos bytes de estado, usted puede decir que el álgebra de Boole le ha proporcionado las respuestas correctas.

b. Convierta los bytes de estado hex en código binario,

27 = 00100111 (byte de estado previo)
63 = 01100011 (byte de estado actual)

Realice una operación O-Exclusiva.

$$00101111 \oplus 01100011 = 01001100$$

Utilice el resultado para realizar una operación AND con el byte de estado previo,

$$00101111 \bullet 01001100 = 00001100$$

Así, podemos tomar la conclusión de que los dispositivos de medida de nivel de líquido, caudal y velocidad han cambiado de estado. Los dispositivos de medida de caudal y de velocidad han pasado de 1 a 0.
Complemente el resultado de la operación AND.

$$\overline{00001100} = 11110011$$

AND este resultado con el resultado de la operación XOR inicial,

$$11110011 \bullet 01001100 = 01000000$$

Así, el dispositivo de medida de nivel de líquido pasó de 0 a 1.

c. Convierta los bytes de estado hex a código binario,

02 = 00000010 (byte de estado previo)
07 = 00000111 (byte de estado actual)

Realice una operación XOR,

$$00000010 \oplus 00000111 = 00000101$$

Utilice este resultado y realice una operación AND con el byte de estado previo,

$$00000010 \bullet 00000101 = 00000000$$

Los dispositivos de medida de velocidad y presión cambiaron de estado. Ninguno de ellos ha pasado de 1 a 0.
Complemente el resultado de la operación AND.

$$\overline{00000000} = 11111111$$

AND este resultado con la operación inicial XOR,

$$11111111 \bullet 00000101 = 00000101$$

Los dispositivos de medida de velocidad y presión pasaron de 0 a 1.

d. Convierta el byte de estado hex en código binario,

A7 = 10100111 (byte de estado previo)
DB = 11011011 (byte de estado actual)

Realice una operación XOR,

$$10100111 \oplus 11011011 = 01111100$$

Utilice este resultado y realice una operación AND con el byte de estado previo,

$$01111100 \bullet 10100111 = 00100100$$

Los dispositivos de medida de nivel de líquido, corriente, tensión, caudal y velocidad pasaron desde 1 a 0.
Complemente el resultado de la operación AND,

$$\overline{00100100} = 11011011$$

AND este resultado con el resultado de la operación XOR inicial

$$11011011 \bullet 01111100 = 01011000$$

Los dispositivos de medida de nivel de líquido, tensión y caudal pasaron desde 0 a 1.

10

Instrucciones de manipulación de bit, rotación y desplazamiento

En este capítulo, examinaremos dos grupos de instrucciones que enriquecen significativamente el conjunto de instrucciones del Z-80, el grupo de *MANIPULACION DE BIT* y el grupo de *ROTACION Y DESPLAZAMIENTO.* Las instrucciones en el grupo de manipulación de bit le permitirán hacer un test y/o cambiar los valores de células de los registros o memoria al nivel individual de bit. Actualmente, las instrucciones de manipulación de bit comprenden cerca del 50% de las nuevas instrucciones del Z-80 que no están disponibles en los sistemas basados en el 8080.

OBJETIVOS

Al completar este capítulo, usted será capaz de hacer lo siguiente:

- Utilizar las instrucciones de manipulación de bit, BIT, SET y RESET.
- Utilizar las instrucciones de rotación y desplazamiento.
- Entender porqué son útiles cada una de las instrucciones de manipulación de bit, rotación y desplazamiento.
- Entender la aplicación de las instrucciones RRD y RLD para procesar números bcd.

PROCESO DE LAS INSTRUCCIONES BIT SET, TEST Y RESET

Un ejemplo simple y universal de la utilización y proceso de las instrucciones *set, reset* y *test de bit* es el procedimiento seguido por los carteros de correo rural y los habitantes para facilitar la entrega del correo. Cada caja de correos, parecida a la de la figura 10-1, en una ruta de correos rural de los Estados Unidos tiene un

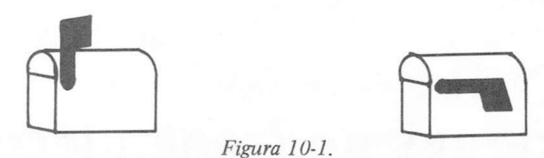


Figura 10-1.

indicador rojo adosado a ella el cual puede estar en posición alta, indicador a uno (estado lógico 1). Si el residente desea que el cartero se pare en su caja de correos para recoger algunas cartas, el residente levanta el indicador a la posición alta. El cartero pasa por allí y mira (TEST del indicador) para ver si el indicador está a 1 (levantado). Si el indicador está a 1, el cartero se para, recoge las cartas y a continuación baja el indicador (RESET) de forma que el proceso pueda ser repetido con la misma convención el próximo día.

Si el indicador no está colocado, el cartero no "atiende" la caja de correos hasta que tenga algunas cartas para entregar al residente. Así, sin la convención de colocar o quitar el indicador, el único tiempo en que el residente podría enviar una carta sería en los días en los que el residente recibía letras y que el cartero tenía que pararse de todas formas. Esta sería una situación incómoda para aquellos que nunca recibimos ninguna carta.

Se está transmitiendo bastante información del residente al cartero con esta convención de un indicador. La razón de que este procedimiento funciona es, desde luego, debido a que existe un acuerdo entre el residente y el cartero como son las condiciones bajo las cuales cada uno debe colocar a 1 (SET) el indicador, o colocarlo a cero (RESET). Este es un ejemplo de un protocolo muy simple. Según Webster, un *protocolo* puede definirse como un código que mantiene una observancia estricta a una etiqueta correcta y a una prioridad. La noción de protocolo será muy importante cuando usted empiece a hacer interfaces entre microcomputadores y otros dispositivos y a otros microcomputadores.

El conjunto de instrucciones del Z-80 de la tabla 10-1 incluye 240 instrucciones diferentes de manipulación de bit:

80 instrucciones de poner a uno un bit ("SET")
80 instrucciones de poner a cero un bit ("RESET"), y
80 instrucciones para hacer un test a un bit ("BIT").

Tabla 10-1. Grupo de instrucciones de manipulación de bit

		DIRECCIONAMIENTO POR REGISTRO							REG. INDIR.	INDEXADO	
	BIT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	GB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Cortesía Zilog, Inc.

Por ejemplo la instrucción de poner a uno un bit, SET 0,A (código de operación: CB C7), coloca un 1 lógico en el bit número cero del acumulador. Obsérvese que el número que especifica el bit (0 en este ejemplo) no es un dato o byte de dirección y no está seguido por una H significando hexadecimal, o por . significando decimal. Considere los dos ejemplos siguientes:

Ejemplo 1

	Acumulador
Antes de la ejecución de SET 0,A	1 1 0 0 1 0 0 0
Después de la ejecución de SET 0,A	1 1 0 0 1 0 0 1

Ejemplo 2

Antes de la ejecución de SET 0,A	1 1 1 1 0 0 1 1
Después de la ejecución de SET 0,A	1 1 1 1 0 0 1 1

Recuerde que los ocho bits en cualquier registro o posición de memoria están siempre numerados desde la izquierda a la derecha empezando con el bit cero y acabando con el séptimo bit.

Una forma corriente de describir la instrucción SET 0,A consiste en decir que coloca el bit cero del acumulador a 1. Una forma más corta y popular de decir esto es decir que la instrucción "SET" el bit cero del acumulador. El uso corriente ha determinado que la frase "SETTING" un bit implica que el valor del bit es 1 después de que se haya ejecutado la instrucción. Similarmente, la frase "RESETTING" un bit significa que el valor del bit es 0 después que se ha ejecutado la instrucción "RESET".

Se debe observar que las instrucciones SET bit y RESET bit son independientes del valor original del bit. Así, un bit que ya estaba previamente "SET" puede ser "SET" de nuevo sin que se produzca un cambio apreciable en el bit. La instrucción, RES 0,D (código de operación: CB 82), pondrá a cero (RESET) el bit cero del registro D.

Ejemplo 3

	Registro D
Antes de la ejecución de RES 0,D	1 1 0 0 1 1 1 1
Después de la ejecución de RES 0,D	1 1 0 0 1 1 1 0

Ejemplo 4

Antes de la ejecución de RES 0,D	0 0 0 0 0 0 0 0
Después de la ejecución de RES 0,D	0 0 0 0 0 0 0 0

Ni las instrucciones "SET" ni las instrucciones "RESET" afectan a ninguno de los indicadores.

Una instrucción algo más complicada es la instrucción test bit. Por ejemplo, la instrucción BIT 0,A (código de operación: CB 47), hace un test al bit cero del acumulador. Si el bit cero del acumulador es un cero entonces está instrucción pondrá

a 1 el indicador de cero. Es decir, el valor del indicador de cero será 1 si el bit cero del acumuldor es cero.

Ejemplo 5

	Acumulador	Indicador de cero
Antes de la ejecución de BIT 0,A	1 0 0 1 1 1 0 0	X
Después de la ejecución de BIT 0,A	1 0 0 1 1 1 0 0	1

Ejemplo 6

Antes de la ejecución de BIT 0,A	1 1 1 0 0 1 1 1	X
Después de la ejecución de BIT 0,A	1 1 1 0 0 1 1 1	0

Ejemplo 7

Antes de la ejecución de BIT 0,A	1 1 1 1 1 1 1 1	X
Después de la ejecución de BIT 0,A	1 1 1 1 1 1 1 1	0

(X = No importa)

Nótese que en ningún caso, BIT 0,A, cambia el valor del acumulador.

Usted encontrará que la instrucción test bit es muy útil. Por ejemplo esta instrucción le permite determinar el valor de un bit particular de cualquier registro o posición de memoria sin tener que crear una máscara mediante un byte y alterar el contenido del acumulador. Considere la siguiente secuencia de dos instrucciones, que realizan la misma función, es decir determinar si el bit D4 de la posición de memoria 01FF está a 1 lógico o a 0 lógico:

Secuencia 1:

```
LD A,(01FFH)
AND 08H
```

Secuencia 2:

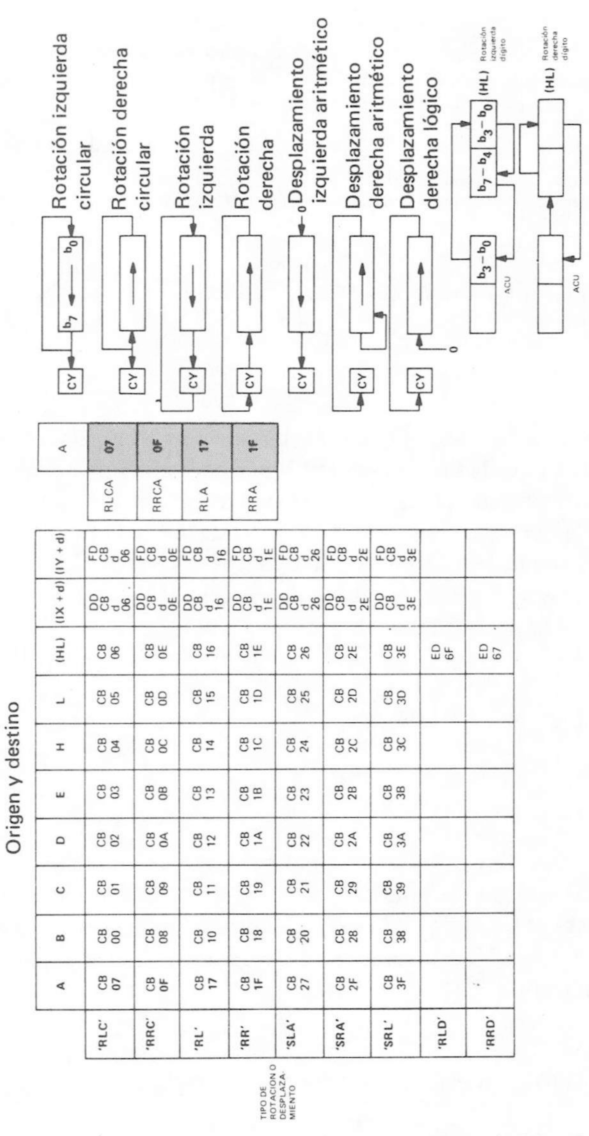
```
LD A,(01FFH)
BIT 4,A
```

La secuencia 1 utiliza el byte 08 como máscara para cambiar el contenido del acumulador colocándose el indicador de cero de acuerdo con el resultado. La secuencia 2 afecta al indicador de cero de la misma forma en lo que hace la secuencia 1 sin tener que cambiar el contenido del registro A. Ambas secuencias necesitan cinco bytes de memoria.

GRUPO DE INSTRUCCIONES DE ROTACION Y DESPLAZAMIENTO

Existen 74 instrucciones de rotación y desplazamiento para el Z-80, que se presentan en la tabla 10-2. Las cuatro instrucciones de rotación del 8080A que son

Tabla 10-2. Rotaciones y desplazamientos



Tipo de rotación o desplazamiento	A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

Cortesía Zilog, Inc.

compatibles RLCA, RRCA, RLA y RRA son *casi* redundantes. Están incluidas en el conjunto de instrucciones del Z-80 solamente para mantener la compatibilidad con el conjunto de instrucciones del Intel 8080A. Se debe observar que estas cuatro instrucciones solamente afectan al indicador de arrastre. La ejecución de cualquiera de estas cuatro instrucciones no afecta a los indicadores de cero, paridad/sobrepasamiento, o indicador de signo. Sin embargo, todas las demás 70 instrucciones de rotación afectan a todos los indicadores de arrastre, cero, paridad/sobrepasamiento, y signo. Además de estas 74 instrucciones, existen dos instrucciones muy especiales de dígito-decimal que discutiremos separadamente.

INSTRUCCIONES DE ROTACION

En primer lugar vamos a considerar las cuatro clases de instrucciones de rotación. Para cada instrucción daremos varios diagramas para ilustrar la operación de rotación particular asociada con el grupo en cuestión.

Rotación izquierda circular (RCL) (figura 10-2)

Podemos interpretar este diagrama examinando el contenido del bit de arrastre y el acumulador antes y después de la ejecución de la instrucción RLC A. Así:

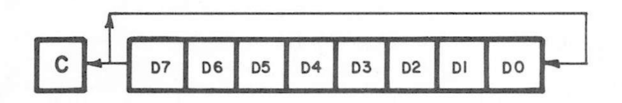


Figura 10-2.

	Bit de arrastre	Acumulador
Antes de la ejecución de RLC A	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de RLC A	D7	D6 D5 D4 D3 D2 D1 D0 D7

Por ejemplo:

Antes de la ejecución de RLC A	X	1 1 0 1 1 1 0 0
Después de la ejecución de RLC A	1	1 0 1 1 1 0 0 1
(X = no importa)		

Rotación derecha circular (RRC) (figura 10-3)

Este diagrama implica que se efectúan los siguientes cambios en el bit de arrastre y en el registro C durante la ejecución de la instrucción RRC C:

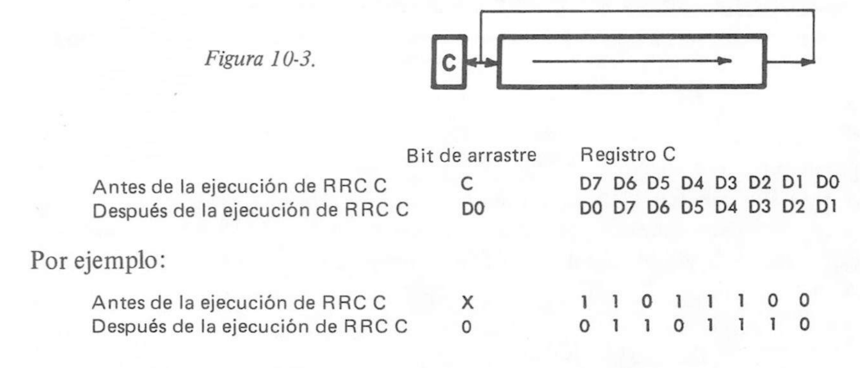


Figura 10-3.

	Bit de arrastre	Registro C
Antes de la ejecución de RRC C	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de RRC C	D0	D0 D7 D6 D5 D4 D3 D2 D1

Por ejemplo:

Antes de la ejecución de RRC C	X	1 1 0 1 1 1 0 0
Después de la ejecución de RRC C	0	0 1 1 0 1 1 1 0

Obsérvese que para las dos instrucciones RRL y RRC el contenido original del indicador de arrastre queda destruido por la ejecución de la instrucción.

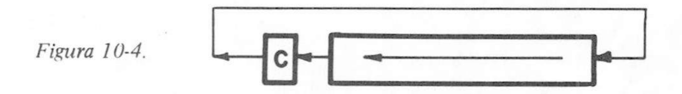


Figura 10-4.

Rotación izquierda (figura 10-4)

Este diagrama implica que se efectúan los siguientes cambios en el bit de arranque y en el acumulador para la instrucción RL A.

	Bit de arrastre	Acumulador
Antes de la ejecución de RL A	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de RL A	D7	D6 D5 D4 D3 D2 D1 D0 C

Por ejemplo:

Antes de la ejecución de RL A	0	1 1 0 1 1 1 0 0
Después de la ejecución de RL A	1	1 0 1 1 1 0 0 0

Rotación derecha (RR) (figura 10-5)

Este diagrama implica que se efectúan los siguientes cambios en el bit de arrastre y en el registro D para la instrucción RR D:

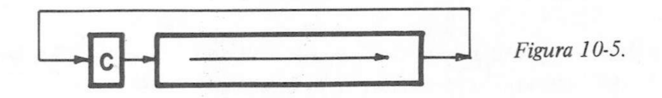


Figura 10-5.

	Arrastre	Registro D
Antes de la ejecución de RR D	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de RR D	D0	C D7 D6 D5 D4 D3 D2 D1

Por ejemplo:

Antes de la ejecución de RR D	0	1 1 0 1 1 1 0 0
Después de la ejecución de RR D	0	0 1 1 0 1 1 1 0

Las cuatro instrucciones de rotación se utilizan frecuentemente para examinar bits sucesivos de un determinado registro o posición de memoria. Por ejemplo, si usted desea encontrar el bit de mayor orden que no es cero en el acumulador, usted puede realizar varias instrucciones sucesivas "rotación izquierda" hasta que el indicador de arrastre se pone a 1:

```
        LD A,X          ; Cargar el acumulador con el byte X a testar
        LD C,08H        ; Contador de bit
CHECK:  DEC C           ; Actualizar el contador de bit
        JP M,END        ; Todos los bytes ¿comprobados?
        RL A            ; Desplazar el próximo bit más significativo al C (arrastre)
        JP NC,CHECK     ; ¿Es 0 o 1? Si 0, entonces probar el próximo bit
END:    RST 38H         ; De lo contrario, devolver el control al sistema operativo
```

En la secuencia de instrucciones precedente, el registro C devuelve el número (7 para el bit más significativo, . . . , 0 para el bit menos significativo) del bit de mayor orden que no es cero del acumulador. Nótese que se puede utilizar la instrucción RLA de un byte en lugar de la instrucción de dos bytes RL A dada previamente debido a que solamente es importante el indicador de arrastre para la lógica del programa.

Otra aplicación interesante de la instrucción de rotación puede verse en el siguiente ejemplo. Suponga que hay ocho procesos que se deben realizar en secuencia, por ejemplo el Proceso 1 primero, Proceso 2 segundo, etc. Sin embargo, no siempre se realizan todos los procesos. Por ejemplo, en algunas situaciones, la secuencia apropiada del proceso es:

Proceso 1		Proceso 5		Proceso 3
Proceso 3	O	Proceso 7	O	Proceso 6
Proceso 5				Proceso 8
Proceso 8				

o cualquier otra secuencia entre 256 secuencias posibles. Para implementar la lógica del programa necesaria para tratar esta situación, se puede utilizar una instrucción de rotación a la derecha.

Primero, se desarrollan las subrutinas para cada proceso, SUB1 para el proceso 1, . . ., SUB8 para el proceso 8. Entonces cada vez que se debe realizar una secuen-

cia de procesos, se genera una descripción bit a bit de la secuencia colocando el bit 0 a 1 si la secuencia 1 se debe realizar, colocando el bit 1 si el proceso 2 se debe realizar, colocando a 1 el bit 7 si se debe realizar el proceso 8. Si un bit está en el nivel lógico 0, el proceso que le está asociado no se realiza. La lógica de bifurcación para realizar cualquier secuencia de proceso así especificada, se puede realizar de la forma siguiente: (Obsérvese que la instrucción RRA es casi idéntica a la instrucción RR.)

```
RRA
CALL C,SUB1
RRA
CALL C,SUB2
RRA
CALL C,SUB3
RRA
CALL C,SUB4
RRA
CALL C,SUB5
RRA
CALL C,SUB6
RRA
CALL C,SUB7
```

Suponemos que el byte que define la secuencia ha sido cargado en el acumulador. Obsérvese que la anterior secuencia de instrucciones utiliza la instrucción de un byte RRA en lugar de la instrucción de dos bytes RRA porque solamente el indicador de arrastre es importante para la lógica del programa.

El próximo grupo de instrucciones que discutiremos es llamado el *GRUPO DE DESPLAZAMIENTO.* Las instrucciones de rotación y desplazamiento se utilizan a menudo conjuntamente unas con las otras para realizar importantes funciones de programación. Discutiremos varios de estos ejemplos en éste y en los próximos capítulos.

INSTRUCCIONES DE DESPLAZAMIENTO

Desplazamiento a la izquierda aritmético (SLA) (figura 10-6)

Este diagrama implica que se efectúan los siguientes cambios en el bit de arrastre y en el acumulador.

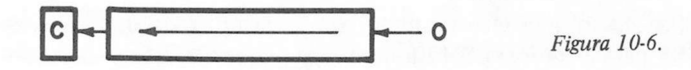


Figura 10-6.

	Bit de arrastre	Acumulador
Antes de la ejecución de SLA A	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de SLA A	D7	D6 D5 D4 D3 D2 D1 D0 0

Esta instrucción tiene el efecto de multiplicar el contenido del acumulador por 2 proporcionando una señal de sobrepasamiento cuando el bit de arrastre es colocado a 1. Considere los siguientes ejemplos:

Ejemplo 1

	Bit de arrastre	Acumulador	
Antes de la ejecución de SLA A	X	0 0 1 1 0 0 1 1	=51 (decimal)
Después de la ejecución de SLA A	0 (sin sobrepasamiento)	0 1 1 0 0 1 1 0	=102 (decimal)

Ejemplo 2

Antes de la ejecución de SLA A	X	0 1 1 0 0 1 1 0	=102 (decimal)
Después de la ejecución de SLA A	0 (sin sobrepasamiento)	1 1 0 0 1 1 0 0	=204 (decimal)

Ejemplo 3

Antes de la ejecución de SLA A	X	1 0 0 1 1 0 0 0	=152 (decimal)
Después de la ejecución de SLA A	1 (sobrepasamiento)	0 0 1 1 0 0 0 0	=48 (decimal)

En los primeros dos ejemplos, el contenido del acumulador fue doblado al aplicar la instrucción SLA A. Pero en el ejemplo 3, pasó de 152 a 48, que dista mucho de doblar el valor inicial. La razón de esto es que 2 veces 152 es igual a 304, el cual es mayor que 256, que es el mayor número positivo que se puede representar por medio de 8 bits (utilizando el sistema binario, y no la representación en complemento a dos). Cuando el número original que se está desplazando es mayor que 128, el resultado del desplazamiento no será dos veces el número (sino el número multiplicado por dos menos 256) y el bit de arrastre es colocado a 1 para indicar esta circunstancia.

La instrucción SLA puede también ser utilizada conjuntamente con la instrucción RL para realizar desplazamientos hacia la izquierda de varios bytes. La siguiente secuencia de instrucciones desplaza el par de registros DE hacia la izquierda de un bit, colocando a cero el bit vacío:

```
SLA E
RL  D
```

Nótese que nosotros también podemos desplazar hacia la izquierda pares de bytes de memoria. Para desplazar hacia la izquierda y colocar a cero el bit libre en las posiciones de memoria 0100 y 0101, podemos utilizar la siguiente secuencia de instrucciones.

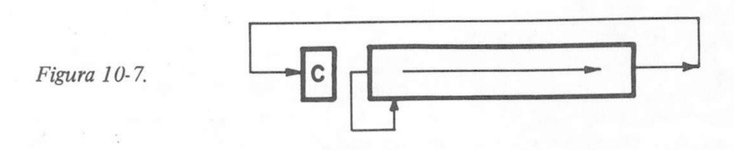
```
LD IX,0100H
SLA (IX)
RL (IX+01H)
```

Los desplazamientos multibyte son muy importantes para la programación de operaciones aritméticas multibyte tales como la multiplicación y la división.

Desplazamiento hacia la derecha aritmético (SRA) (figura 10-7)

Este diagrama implica que se efectúan los siguientes cambios en el bit de arrastre y en el acumulador.

Figura 10-7.



	Bit de arrastre	Acumulador
Antes de la ejecución de SRA A	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de SRA A	D0	D7 D7 D6 D5 D4 D3 D2 D1

Obsérvese que el valor del bit de arrastre queda destruido.

Esta instrucción tiene el efecto de dividir el contenido del registro dado por 2, colocando el "resto" de esta división en el bit de arrastre. Se lleva a cabo la división en complemento a dos, es decir, la instrucción supone que el número que está en el registro está en complemento a dos y produce un cociente en el mismo registro en forma de complemento a dos. Considere los siguientes ejemplos:

Ejemplo 4

	Bit de arrastre	Acumulador
Antes de la ejecución de SRA A	X	0 0 0 0 1 1 1 1 = 15 (decimal)
Después de la ejecución de SRA A	1 (resto)	0 0 0 0 0 1 1 1 = 7 (decimal)

Ejemplo 5

Antes de la ejecución de SRA A	X	1 0 0 0 1 1 1 0 = −114 (decimal)
Después de la ejecución de SRA A	0 (resto)	1 1 0 0 0 1 1 1 = −57 (decimal)

La instrucción SRA se puede utilizar conjuntamente con la instrucción RR para realizar desplazamientos hacia la derecha multibits de una forma muy parecida a como las instrucciones SLA y RL realizan desplazamientos hacia la izquierda. Por ejemplo se puede desplazar hacia la derecha el par de registros HL, con el bit D7 del registro H colocado en el bit vacío, de la siguiente forma:

```
SRA H
RR  L
```

Desplazamiento hacia la derecha lógico (SRL) (figura 10-8)

	Bit de arrastre	Acumulador
Antes de la ejecución de SRL A	C	D7 D6 D5 D4 D3 D2 D1 D0
Después de la ejecución de SRL A	D0	0 D7 D6 D5 D4 D3 D2 D1

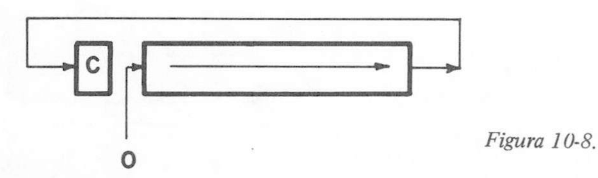


Figura 10-8.

Esta instrucción tiene el efecto de dividir el número situado en el registro 2, cuando el número que está en el registro se considera que es un número binario positivo de 8 bits. El cociente aparece en el registro después de la ejecución de la instrucción y el resto aparece en el bit de arrastre.

Ejemplo 6

	Bit de arrastre	Acumulador
Antes de la ejecución de SRL A		1 0 0 0 0 0 1 1 = 131 (decimal)
Después de la ejecución de SRL A	(resto)	0 1 0 0 0 0 0 1 = 65 (decimal)

Para efectuar secuencias de desplazamientos a la derecha multibits, colocando a cero el bit de más peso, usted puede utilizar las instrucciones SRL y RR. Por ejemplo, para efectuar un desplazamiento hacia la derecha, colocando el bit de más peso a cero, una secuencia de 8 bytes situados en las posiciones de memoria hasta 0100 a 0107 usted puede utilizar la siguiente secuencia de instrucciones:

```
        LD B,08H
        LD HL,0107H
        SRL (HL)
SHIFT:  DEC HL
        DEC B
        JP Z, END
        RR (HL)
        JP SHIFT
END:    RST 38H
```

Vamos a discutir las dos instrucciones especiales de rotación de dígito-decimal. Primero describiremos las instrucciones RLD y RRD y daremos ejemplos de como se pueden utilizar.

Rotación izquierda decimal (RLD) (figura 10-9)

	Acumulador	Célula de memoria señalada por HL
Antes de la ejecución de RLD	D7 D6 D5 D4 D3 D2 D1 D0	B7 B6 B5 B4 B3 B2 B1 B0
Después de la ejecución de RLD	D7 D6 D5 D4 B7 B6 B5 B4	B3 B2 B1 B0 D3 D2 D1 D0

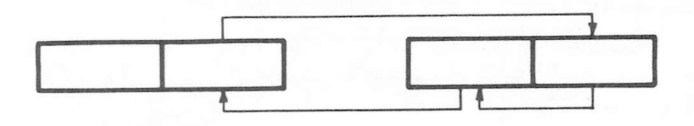


Figura 10-9.

Por ejemplo:

Antes de la ejecución de RLD	1 1 0 1 1 0 0 0	0 0 1 0 1 1 0 0
Después de la ejecución de RLD	1 1 0 1 0 0 1 0	1 1 0 0 1 0 0 0

Observe que esta instrucción efectúa una rotación de la mitad de un byte (*NIBBLES*) de una vez. Y no hace solamente esto. En otras palabras, los cuatro bits 1000 en el "nibble" de menor orden del acumulador aparecen en su destino final como 1000, no como 0001.

Rotación decimal derecha (RRD) (figura 10-10)

	Acumulador	Célula de memoria señalada por HL
Antes de la ejecución de RRD	D7 D6 D5 D4 D3 D2 D1 D0	B7 B6 B5 B4 B3 B2 B1 B0
Después de la ejecución de RRD	D7 D6 D5 D4 B3 B2 B1 B0	D3 D2 D1 D0 B7 B6 B5 B4

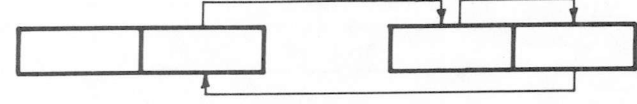


Figura 10-10.

Por ejemplo:

Antes de la ejecución de RRD	1 1 0 1 0 0 0 1	1 0 1 1 1 1 1 0
Después de la ejecución de RRD	1 1 0 1 1 1 1 0	0 0 0 1 1 0 1 1

Las instrucciones RLD y RRD son especialmente útiles para procesamiento de datos que incluyen nibbles (medio byte) en lugar de bits y bytes. Los programas que utilizan la representación bcd (binario eodificado en decimal) para los números son excelentes ejemplos de este tipo de procesamiento orientado al tratamiento

de medios bytes (nibbles). Recuerde que la representación de un número en bcd hace corresponder cuatro bits a un número decimal, permitiendo así, dos números decimales por byte. Por ejemplo la representación bcd para el número decimal 83 es

1 0 0 0	0 0 1 1
nibble 1=8	nibble 2=3

Inversamente, la representación bcd del byte:

0 1 1 1 0 0 0 0

es el número decimal 70. Así en términos de representación bcd la instrucción RLD efectúa una rotación hacia la izquierda de números decimales entre la célula de memoria señalada por HL y el acumulador. Similarmente la instrucción RRD efectúa una rotación de dígitos bcd hacia la derecha.

Por ejemplo, la siguiente secuencia de instrucciones efectuará un desplazamiento hacia la izquierda colocando a cero el bit libre una secuencia de ocho dígitos bcd situados en las cuatro posiciones de memoria en secuencia desde 0100 hasta 0103:

```
        LD B,04H
        LD HL,0100H
        LD A,00H
AGAIN:  RLD
        INC HL
        DEC B
        JP NZ,AGAIN
        RST 38H
```

Obsérvese que el dígito decimal de mayor orden (el nibble de mayor orden de la posición de memoria 0103) se deja en el nibble de menor orden del acumulador.

El nibble de mayor orden del acumulador no queda afectado por esta operación. El siguiente diagrama ilustra los efectos de la rutina anterior en el acumulador y en las posiciones de memoria 0100 hasta 0103.

	Acumulador	0103	0102	0101	0100
Antes:	X X	8 7	6 5	4 3	2 1
Después:	0 8	7 6	5 4	3 2	1 0

En el experimento N.° 2 se estudia una aplicación adicional para estas instrucciones de rotación, que efectúa un cambio de una representación en bcd a ASCII.

INTRODUCCION A LOS EXPERIMENTOS

Los siguientes experimentos están diseñados para ayudarle a comprender como trabajan las instrucciones de manipulación de bits, rotación y desplazamiento y para mostrarle sus aplicaciones. En el experimento N.° 1, le introducimos en la representación ASCII de los números. En cada uno de estos experimentos, le damos programas que convierten números entre tres formas de representación: binario, bcd y ASCII.

Experimento N.°	Comentarios
1	Demuestra la utilización de las instrucciones BIT y RR en un programa que pasa de una representación binaria a ASCII.
2	Demuestra la utilización de la instrucción RL en un programa para convertir ASCII a representación binaria.
3	Demuestra la utilización de la instrucción RRD en un programa para convertir bcd a representación ASCII.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar la utilización de las instrucciones BIT en un programa para convertir binario a una representación ASCII. La representación ASCII de los números de 0 al 9 está dada en la siguiente tabla:

N.° DECIMAL	REPRESENTACION ASCII EN HEX
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39

ASCII no es más que otro método para representar números utilizando bytes de 8 bits. Para muchos trc (pantallas) e impresoras, el ASCII es el código estándar para representar números, letras y caracteres especiales tales como ;, !, <, >, ?, etc. La conversión de binario a ASCII consiste en entrar series de caracteres de 0 y 1 y sacar la correspondiente secuencia de 30 y 31. Por ejemplo, el byte 0 1 0 0 1 1 0 1 sería convertido a la siguiente serie de grupos de ocho bits (representado en hex):

30 31 30 30 31 31 30 31

El siguiente programa convierte el byte situado en el registro B a una serie de grupos de ocho bits guardados en las posiciones de memoria empezando en la posición 0200.

Programa N.° 29

Posición de memoria	Código objeto		Código fuente	Comentarios
0100	0E 08		LD C,08H	; Contador de bit
0102	21 00 02		LD HL,0200H	; Inicio del ASCII
0105	36 30	NXTBIT:	LD (HL),30H	; Cero ASCII
0107	CB 40		BIT 0,B	; Test del bit
0109	28 01		JR Z,ZERO	; Si no incrementar
010B	34		INC (HL)	; pasar de 30 a 31
010C	23	ZERO:	INC HL	; Incrementar el indicador de caracteres ASCII
010D	CB 18		RR B	; Desplazar B para mirar al próximo bit
010F	0D		DEC C	; Actualizar el contador de bit
0110	20 F3		JR NZ,NXTBIT	; ¿Se han trasladado todos los bits?
0112	FF		RST 38H	

Paso 1

Cargar el programa anterior en las direcciones indicadas. Ejecutarlo utilizando diferentes bytes de muestra en el registro B. Por ejemplo, si el contenido del registro B es

0 1 1 1 1 0 0 0

entonces las posiciones de memoria desde 0200 hasta 0207 deben contener (en hex)

(0200)=30
(0201)=30
(0202)=30
(0203)=31
(0204)=31
(0205)=31
(0206)=31
(0207)=30

Paso 2

Reemplace el programa precedente utilizando una instrucción de rotación y el indicador de arrastre para trasladar cada bit de binario a ASCII. Compruebe y depure su programa para asegurarse de que trabaja perfectamente.

EXPERIMENTO N.° 2

Propósito

El propósito de este experimento es el de demostrar la utilización de la instrucción RL en un programa para convertir ASCII a representación binaria. En este programa, se entra un grupo de ocho ceros ASCII (30 hex) y unos (31 hex), y se saca un solo byte con los bits adecuados colocados a 0 lógico o a 1. El siguiente programa convierte el grupo de ocho bytes empezando en la posición 0200 a un solo byte que está contenido en el registro B.

Programa N.° 30

Posición de memoria	Código objeto		Código fuente	Comentarios
0120	0E 08		LD C,08H	; Contador de bit
0122	21 00 02		LD HL,0200H	; Dirección de inicio del ASCII
0125	7E	NXT:	LD A, (HL)	; Obtener el byte
0126	1F		RRA	; Mover el bit 0 al indicador de arrastre C
0127	CB 18		RR B	; Desplazar el indicador de arrastre C al registro B
0129	23		INC HL	; Señalar al próximo byte
012A	0D		DEC C	; Actualizar el contador de bit
012B	20 F8		JR NZ,NXT	; ¿Hemos mirado todos los bytes?
012D	FF		RST 38H	

Paso 1

Cargar y ejecutar el programa anterior utilizando varios grupos de caracteres ASCII de muestra como entrada.

Paso 2

Volver a escribir el programa para procesar el grupo de bytes ASCII desde la posición 0207 hacia atrás a la 0200. ¿Cómo influenciará esto a las instrucciones de

rotación? Comprobar y depurar este programa asegurándose de que funciona correctamente.

EXPERIMENTO N.° 3

Propósito

El propósito de este experimento es el de demostrar la utilización de la instrucción RRD en un programa para convertir bcd a representación ASCII. En este programa, la conversión de ASCII a bcd significa que se entran una serie de bytes "bcd empaquetado", es decir bytes que contienen dos números de cuatro bits, y se saca una serie de números ASCII, uno por byte. Por ejemplo,

Byte 1	0 0 1 1	1 0 0 1	=39 (bcd)
Byte 2	0 1 0 1	1 0 0 0	=58 (bcd)
Byte 3	0 0 0 0	0 0 0 1	=01 (bcd)
Byte 4	0 1 1 1	0 0 0 0	=70 (bcd)

se convierte a una serie de ASCII de ocho bytes (escrito en hex)

33 39 35 38 30 31 37 30

El siguiente programa convierte una serie de bytes empaquetados en bcd, cuya dirección de inicio está contenida en el par de registros HL, a una serie de bytes ASCII cuya dirección de inicio está contenida en el par de registros DE. El par de registros HL es colocado en 0210 y los registros DE inicialmente en 0301. El registro C contiene el número de bytes bcd empaquetados que se van a convertir. Hemos colocado el contenido de C a 04.

Programa N.° 31

Posición de memoria	Código objeto	Código fuente	Comentarios
0130	3E 30	LD A,30H	; Inicializar el nibble de mayor orden
			; del acumulador a 3
0132	21 10 02	LD HL,0210H	; Dirección de origen (bcd empaquetado)
0135	11 01 03	LD DE,0301H	; Dirección de destino (serie de caracteres
			; ASCII)
0138	0E 04	LD C,04H	; Número de bytes de origen
013A	ED 67	BCD: RRD	; Desplazar el nibble de menor peso
			; al acumulador. Puesto que el nibble

```
                                       ; de mayor orden es 3 obtenernos el
                                       ; equivalente ASCII
0130    12        LD (DE),A            ; Guardar el byte ASCII
013D    1B        DEC DE               ; Decrementar el indicador de destino
                                       ; para el número bcd de mayor peso
013E    ED 67     RRD                  ; Desplazar el nibble de mayor peso
                                       ; al acumulador
0140    12        LD (DE),A            ; Guardar el caracter ASCII
0141    ED 67     RRD                  ; Desplazar el nibble de mayor peso
                                       ; de nuevo al byte de origen restau-
                                       ; rándolo a su forma inicial
0143    13        INC DE               ; Actualizar el indicador de destino
0144    13        INC DE
0145    13        INC DE
0146    23        INC HL               ; Actualizar el indicador de origen
0147    0D        DEC C                ; Actualizar el contador de bytes de origen
0148    20 F0     JR NZ,BCD            ; ¿Hemos terminado?
014A    FF        RST 38H
```

Paso 1

Cargar el programa anterior en las direcciones indicadas. Ejecutar el programa en modo paso a paso para varias muestras de series de números bcd empaquetados para tratar de entender como funciona.

Paso 2

Vamos a discutir como funciona el programa anterior. Considere el diagrama de la figura 10-1.

Posición de memoria	Núm. de mayor orden bcd D7 D6 D5 D4	Núm. bcd de menos peso D3 D2 D1 D0
0210	BCD1	BCD2
0211	BCD3	BCD4
0212	BCD5	BCD6
0213	BCD7	BCD8

0300	ASCII1	cargado después del 2.° RRD
0301	ASCII2	cargado después del 1.er RRD
0302	ASCII3	cargado después del 5.° RRD
0303	ASCII4	cargado después del 4.° RRD
0304	ASCII5	cargado después del 8.° RRD
0305	ASCII6	cargado después del 7.° RRD
0306	ASCII7	cargado después del 11.° RRD
0307	ASCII8	cargado después del 10.° RRD

	Acumulador	Posición de memoria 0210
Configuración inicial	3 0	BCD1 BCD2
Después de la primera instrucción RRD	3 BCD2	0 BCD1
Nota: 3 BCD2 = ASCII2 es guardado en (0301)		
Después de la segunda instrucción RRD	3 BCD1	BCD2 0
Nota: 3 BCD1 es guardado en (0300)		
Después de la tercera instrucción RRD	3 0	BCD1 BCD2
Nota: Todo ha sido restaurado, el traslado del byte fuente está terminado.		

(Todo restaurado, el traslado del byte fuente está terminado.)

Existen tres hechos cruciales para entender el funcionamiento de este programa:

1. La instrucción RRD trabaja para mover nibbles entre la memoria y el acumulador. Esto se muestra en la figura 10-1.
2. El funcionamiento de la instrucción RRD necesita que BCD2 sea convertido a ASCII *antes* que BCD1 sea convertido a ASCII1. Esto explica el porqué el registro DE es inicializado a 0301, decrementado e incrementado tres veces a continuación.
3. El funcionamiento de la instrucción RRD nos permite inicializar el acumulador a 30 y realizar la conversión de bcd a ASCII simplemente desplazando los dígitos bcd en el nibble de menor peso, dejando el nibble de mayor peso constante.

Obsérvese que este programa se aparta de nuestra práctica normal de asociar los pares mitad de mayor orden con las posiciones de memoria más altas: BCD1, el nibble de mayor orden, es trasladado a ASCII a la posición 0300 y BCD2, el nibble de menor peso, es trasladado a ASCII2 en la posición 0301. Esto es debido al hecho de que normalmente es deseable imprimir el byte de mayor orden primero. Así, la serie de caracteres ASCII que empieza en la posición 0300 se puede sacar en secuencia a una impresora y aparecer así, en el orden de "impresión" normal con los dígitos de mayor peso precediendo a los dígitos de menor peso.

11

Instrucciones aritméticas y de búsqueda de bloques

En este capítulo continuaremos nuestra discusión del grupo de instrucciones aritméticas y lógicas de 8 bits y del grupo de instrucciones de uso general AF. También investigaremos el grupo de instrucciones aritméticas de 16 bits y el potente grupo de instrucciones de búsqueda de bloques. Estos cuatro grupos de instrucciones aparecen en las tablas 11-1, 11-2, 11-3 y 11-4 respectivamente. Al final de este capítulo, usted habrá tenido ocasión de utilizar todas las instrucciones que puede realizar el microprocesador Z-80 excepto las entradas, salidas y las instrucciones relacionadas con las interrupciones que serán cubiertas en detalle en un siguiente volumen.

OBJETIVOS

Al completar este capítulo, usted será capaz de:

- Escribir programas para sumar, restar, multiplicar y dividir números enteros binarios de 8 bits.
- Escribir programas para sumar, restar, multiplicar y dividir números binarios de 16 bits.
- Entender y utilizar la instrucción (DAA) ajuste decimal del acumulador conjuntamente con la aritmética bcd.

Tabla 11-1. Grupo de instrucciones aritméticas y lógicas de 8 bits

ORIGEN

	DIRECCIONAMIENTO POR REGISTRO							REG. INDIR.	INDEXADO		INMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
SUMAR CON ARRASTRE 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
RESTAR 'SUB	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
RESTAR CON ARRASTRE 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARAR 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENTAR 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENTAR 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Cortesía Zilog, Inc.

Tabla 11-2. Operaciones AF de aplicación general

Ajuste Decimal del Ac. 'DAA'	27
Complementar Acu, CPL	2F
Negar Acu. 'NEG' (complemento a dos)	ED 44
Complementar el indicador de arrastre, 'CCF'	3F
Colocar a 1 el indicador de arrastre 'SCF'	37

Cortesía Zilog, Inc.

Tabla 11-3. El grupo aritmético de 16 bits

		ORIGEN					
DESTINO		BC	DE	HL	SP	IX	IY
'ADD'	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
SUMAR CON ARRASTRE Y COLOCAR INDICADORES 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
RESTAR CON ARRASTRE Y COLOCAR INDICADORES 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENTAR 'INC'		03	13	23	33	DD 23	FD 23
DECREMENTAR 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

Cortesía Zilog, Inc.

Tabla 11-4. Grupo de búsqueda de bloques

POSICION DE BUSQUEDA

REG. INDIR. (HL)	
ED A1	**'CPI' Inc HL, Dec BC**
ED B1	'CPIR', Inc HL, Dec BC repetir hasta que BC = 0 o se encuentre igualdad
ED A9	**'CPD' Dec HL & BC**
ED B9	'CPDR' Dec HL y BC Repetir hasta que BC = 0 o se encuentre igualdad

HL señala a la posición de memoria que se ha de comparar con el contenido del acumulador
BC es el contador de bits

Cortesía Zilog, Inc.

- Describir las funciones de los indicadores H y N conjuntamente con la aritmética bcd y la instrucción DAA.
- Entender y utilizar las instrucciones sumar-con-arrastre y restar con arrastre.
- Entender y utilizar las instrucciones de comparación CP, y las potentes instrucciones de búsqueda de bloques que son una extensión de éstas.

GRUPO ARITMETICO DE 8 BITS

El grupo de instrucciones aritméticas de 8 bits comprende todas las operaciones de suma o resta de bytes de ocho bits. Las instrucciones INC y DEC suman o restan 01 hexadecimal de un registro o memoria especificado. Las instrucciones ADD y SUB especifican un byte en un registro o posición de memoria que se ha de sumar o restar a/del byte situado en el acumulador con el resultado suma/diferencia almacenado en el acumulador. Las instrucciones INC, DEC, ADD y SUB para las operaciones con 8 bits afectan todas ellas a los indicadores de estado de la siguiente forma:

Indicador de cero: Si el resultado en el acumulador es cero, entonces el indicador se coloca a 1, de lo contrario a 0.

Indicador de signo: Si el resultado en el acumulador es negativo (es decir, el bit más significativo es 1 lógico), el indicador de signo (S) vale 1, de lo contrario 0.

Indicador de arrastre: Si la operación INC, DEC, ADD o SUB se produce en un arrastre de/desde un 9.° bit "fantasma", el indicador de arrastre se coloca a 1, de lo contrario a 0.

Indicador P/V (paridad/sobrepasamiento): El indicador P/V se comporta exactamente como un indicador de sobrepasamiento en la aritmética de complemento a dos. Ver la discusión sobre esto en el capítulo 8.

Indicador H: El indicador de medio arrastre (H) se coloca a 1 si se produce un arrastre como resultado de sumar los dos dígitos de menor peso de dos números empaquetados bcd, de lo contrario se coloca a 0. Discutiremos extensamente este indicador al estudiar la instrucción DAA.

Indicador N: El indicador de resta (N) se coloca a 1 para todas las operaciones relacionadas con la resta, y se pone a 0 para todas las operaciones relacionadas con la suma. Este indicador se discutirá también extensamente en la sección que trata de la instrucción DAA.

Aquí hay varios ejemplos que ilustran las operaciones INC, DEC, ADD y SUB. Cuando se coloca ● en una columna etiquetada con un indicador, señala que este

indicador no está afectado por la operación, es decir que la instrucción no lo cambia.

Instrucción	Acumulador antes de la ejecución	Acumulador después de la ejecución	Indicadores después de la ejecución (X = no importa)					
			S	**Z**	**H**	**P/V**	**N**	**C**
INC A	04	05	0	0	0	0	0	•
INC A	FF	00	0	1	1	0	0	•
DEC A	00	FF	1	0	1	0	1	•
ADD 80H	00	80	0	0	0	0	0	0
ADD 80H	80	F0	1	0	0	1	0	0
ADD F0H	F0	E0	1	0	0	0	0	1
ADD 11H	22	33	0	0	0	0	0	0
ADD 18H	29	41	0	0	1	0	0	0
ADD 94H	93	27	0	0	0	1	0	1
ADD 99H	99	32	0	0	1	1	0	1
SUB 33H	33	00	0	1	0	0	1	0
SUB 02H	10	0E	0	0	1	0	1	0
SUB 22H	10	EE	1	0	1	0	1	1

Para entender cada uno de los ejemplos anteriores, realice la operación indicada en aritmética binaria y siga las reglas descritas previamente para colocar a 1 y a 0 los indicadores afectados. Por ejemplo considere la instrucción,

ADD 18H

cuando el acumulador contiene 29. Entonces se puede escribir la suma binaria:

```
    0 0 1 0 1 0 0 1    Acumulador
  + 0 0 0 1 1 0 0 0
    ---------------
    0 1 0 0 0 0 0 1
```

Indicador S:	0 porque el bit D7 es cero
Indicador Z:	0 porque el resultado no es cero
Indicador H:	1 porque se produjo un arrastre desde el bit D3 al D4 durante la suma bit a bit
Indicador P/V:	0 porque dos números positivos en complemento a dos sumados dan un número positivo en complemento a dos
Indicador N:	0 debido a que se trata de una operación de suma
Indicador C:	0 porque no se produce arrastre más allá del bit D7

Las instrucciones de 8 bits ADC, *sumar con arrastre* y SBC, *restar con arrastre* realizan una operación de tres pasos:

Paso 1: Sumar o restar el byte indicado de/desde el acumulador como si se realizara una instrucción ADD o SUB. No cambia ningún bit de los indicadores.

Paso 2: Sumar o restar el indicador C de/desde el acumulador. Esto es, si el indicador C estaba previamente a 1 antes de la ejecución de la instrucción ADC o

SBC, sumar o restar 01 del byte situado en el acumulador. Si el indicador C estaba a cero, no se cambia el acumulador.

Paso 3: Ajustar los indicadores basándose en los resultados de los dos pasos anteriores.

Aquí están algunos ejemplos para ilustrar las instrucciones ADC y SBC.

Instrucción	Acumulador antes de la ejecución	indicador C antes de la ejecución	Acumulador después de la ejecución	Indicadores después de la ejecución S	Z	X	H	X	P/V	N	C
ADC 00H	01	1	02	0	0		0		0	0	0
ADC 00H	01	0	01	0	0		0		0	0	0
ADC 90H	97	1	28	0	0		0		1	0	1
ADC 19H	39	1	53	0	0		1		0	0	0
SBC 00H	00	1	FF	1	0		1		0	1	1
SBC 01H	00	1	00	0	1		0		0	1	0
SBC 80H	00	1	7F	1	0		1		1	1	1

Las instrucciones ADC y SBC son especialmente útiles para las operaciones aritméticas multibyte. Considere el siguiente programa que realiza una suma multibyte o de *precisión múltiple* de dos números binarios almacenados en la memoria. El número máximo de bytes se guarda en el registro C. La serie de bytes que representan el primero y segundo sumandos empiezan con sus bytes menos significativos en las posiciones de memoria que señalan los registros HL y IX, respectivamente. IY señala a la dirección de inicio (byte menos significativo) de la serie de bytes que representa la suma.

```
       LD C,08H        ; Ocho bytes en cada sumando
       LD HL,0200H     ; Sumando # 1
       LD IX,0210H     ; Sumando # 2
       LD IY, 0220H    ; Suma
       SUB A           ; Borrar el acumulador y el indicador C
ADDB:  LD A,(HL)       ; Adquirir un byte de la serie # 1
       ADC (IX)        ; Sumar el byte del grupo # 2
       LD (IY),A       ; Guardar la suma en el grupo # 3
       INC HL          ; Actualizar los indicadores de la memoria
       INC IX
       INC IY
       DEC C           ; ¿Hemos procesado todos los bytes?
       JR NZ,ADDB      ; Si no, sumar los próximos bytes
       JR C,ERROR      ; Sí, comprobar si hay arrastre. Sobrepasamiento si C = 1
       RST 38H         ; Retornar el control al sistema operativo
```

Vamos a discutir como funciona este programa. La figura 11-1 muestra los sumandos y la suma en sus respectivas posiciones de memoria y en la forma en que el programa los manipula. Se muestran los valores iniciales para HL, IX y IY. Estos registros se actualizan a medida que se van sumando pares sucesivos de bytes de

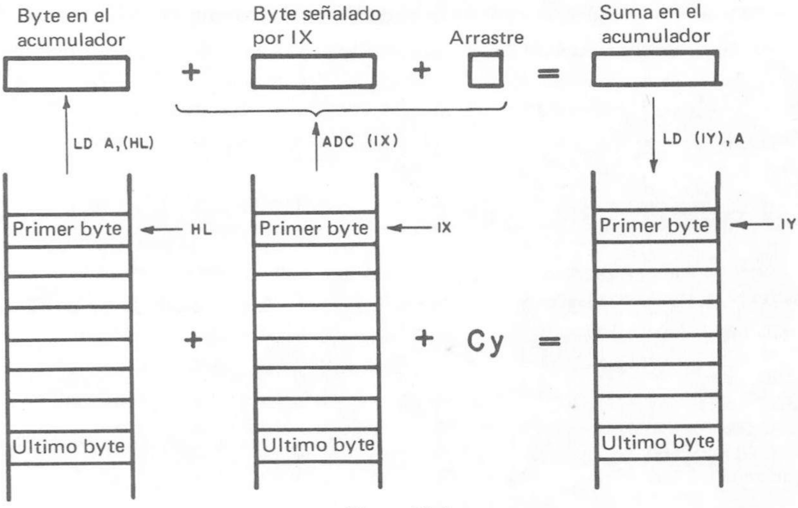


Figura 11-1.

memoria. Obsérvese que el arrastre si hay alguno, es sumado al próximo byte de mayor orden en la instrucción ADC. La instrucción SUB A borra inicialmente el indicador de arrastre C de forma que la primera suma es equivalente a una instrucción ADD. Después de que todos los pares de bytes desde el conjunto n.º 1 y del conjunto n.º 2 han sido sumados juntos, se hace una comprobación para ver si existe un arrastre en la instrucción JP C,ERROR. La instrucción ADC (IX) fue la última instrucción que podía afectar al indicador C de forma que el indicador C todavía representa la existencia o no de un sobrepasamiento de la suma de los dos bytes (más significativos) *ERROR* representa la posición de memoria de una rutina que imprime mensajes de sobrepasamiento, que no se muestra aquí.

El programa precedente es igualmente aplicable a ún suma multibyte en complemento a dos, suponiendo que los dos sumandos sean dos números de n bit en complemento a dos, en donde n=8 X (longitud en bytes de los dos sumandos). Para la suma en complemento a dos el indicador P/V se debe comprobar como indicador de sobrepasamiento, en lugar del indicador C como en el caso anterior.

INSTRUCCION DAA

Para la aritmética binaria codificada en decimal se necesita una instrucción especial para convertir un resultado basado en una operación binaria a resultados en el formato bcd apropiado. El Z-80 solamente conoce un método para la suma y resta, que es el binario. Puesto que la suma y resta en complemento a dos y binaria son

esencialmente lo mismo (exceptuando la detección de sobrepasamiento), el Z-80 puede también realizar aritmética en complemento a dos. La aritmética decimal no es lo mismo que la aritmética binaria. Considere el siguiente problema de suma, solucionado en primer lugar como una suma binaria y después como una suma decimal de dos números bcd empaquetados:

```
 00001000 =  8 (base 10) o 08 (bcd empaquetado)
 00001001 =  9 (base 10) o 09 (bcd empaquetado)
 --------
 00010001 = 17 (base 10) o 11 (bcd empaquetado)
```

Obsérvese que el resultado, interpretado como un número binario, es correcto, pero interpretado como un número bcd empaquetado es incorrecto. La explicación de esto es la diferencia entre la base de los números. El Z-80 trata los dos dígitos de un número bcd empaquetado como dos dígitos *hexadecimales* porque cuatro dígitos en representación binaria pueden representar dieciséis valores distintos. Así durante una operación aritmética como la suma, se produce un arrastre al dígito de la izquierda cuando la suma es mayor que 16. Para una suma bcd este arrastre se debe producir cuando la suma es mayor de 10. Así las sumas binaria y bcd no producen el mismo resultado cuando

a. La suma de dos nibbles de 4 bits está entre 10 y 15 inclusive, por ejemplo:

```
   1001  9
 + 0010  2
   ----
   1011  B como una suma hex, debe ser 11 como una suma bcd empaquetada.
```

En este caso, el binario no produce arrastre al próximo nibble, cuando sí lo hace la suma decimal.

b. La suma de dos nibbles de 4 bits es mayor o igual de 16, por ejemplo:

```
   1001  9
 + 1001  9
   ----
  10010  12 como una suma hex, debe ser 18 como un número bcd empaquetado.
```

En este caso, el binario produce un arrastre al próximo nibble pero se produce seis números "demasiado tarde".

En ambos casos presentados, la respuesta hex menos la respuesta decimal es seis. De esto se deduce que cuando ocurre (a) o (b) esto es *siempre* cierto. Así, la CPU Z-80 tiene una instrucción especial, la instrucción *Ajuste Decimal de Acumulador,* la cual puede detectar cuando se produce (a) o (b) y añadir seis al nibble si es apropiado. El proceso de detección es muy simple. Cuando se produce el caso (a) se obtiene un nibble con un equivalente hex no decimal tal como A, B, C, D, E o F. Así un nibble cuyo valor es mayor de nueve en el resultado indica que se le debe

sumar seis a este nibble. Cuando se produce (b) se detecta mediante un indicador de arrastre colocándose a 1 o bien C o bien H. El indicador H se coloca a 1 si se produce un arrastre como resultado de sumar los dos nibbles de menor peso. El indicador C se coloca a 1 si se produce un arrastre como resultado de sumar los dos nibbles de mayor peso.

Considere la siguiente secuencia de instrucciones que realizan una suma bcd empaquetada entre el contenido del acumulador y del registro B:

```
ADD B
DAA
```

Vamos a ver como actúan estas instrucciones en seis conjuntos diferentes de datos.

	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Grupo 5
Registro B	11	19	91	99	09
Acumulador antes de ADD B	22	18	81	88	05
Después de ADD B:					
Acumulador	33	31	12	21	0E
Indicador H	0	1	0	1	0
Indicador C	0	0	1	1	0
Acumulador después de DAA	33	37	72	87	14

El tercer y cuarto conjunto de valores de los datos (91 y 81; 99 y 88) representan sumas que son mayores que 99, que es el mayor número bcd empaquetado que puede contener el acumulador. Así el 1 final que se debería arrastrar como tercer dígito en el lugar de las centenas se pierde. Obsérvese que 172 y 187 son las respuestas correctas. El sobrepasamiento en tales casos se indica porque el indicador C se pone a 1. El último par de datos (09 y 05) es un ejemplo de que se ha producido la condición precedente (a). No se ha producido ni H ni C pero se necesita un ajuste del nibble de menor orden debido a que E no es un número decimal.

Así para convertir nuestro programa de muestra que sumaba numeros binarios multibyte a un programa que sume número bcd empaquetados multibyte se necesita solamente un simple cambio. Simplemente insertar la instrucción DAA entre las instrucciones ADC (IX) y LD (IY),A.

INSTRUCCIONES ARITMETICAS DE 16 BITS

Las instrucciones de 8 bits ADD, ADC, SBC, INC y DEC tienen análogos de 16 bits los cuales realizan esencialmente las mismas operaciones utilizando pares de registros de 16 bits. Las instrucciones de 16 bits tratan a los indicadores de una

Tabla 11-5. El grupo aritmético de 16 bits

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
ADD HL, ss	HL ← HL+ss	↕	•	•	•	0	X	00 ss1 001	1	3	11	ss Reg.
												00 BC
ADC HL, ss	HL←HL+ss+CY	↕	↕	V	↕	0	X	11 101 101	2	4	15	01 DE 10 HL
								01 ss1 010				11 SP
SBC HL, ss	HL←HL-ss-CY	↕	↕	V	↕	1	X	11 101 101	2	4	15	
								01 ss0 010				
ADD IX, pp	IX ← IX + pp	↕	•	•	•	0	X	11 011 101	2	4	15	pp Reg.
								00 pp1 001				00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY←IY+rr	↕	•	•	•	0	X	11 111 101	2	4	15	rr Reg.
								00 rr1 001				00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6	
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101	2	2	10	
								00 100 011				
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101	2	2	10	
								00 100 011				
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6	
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101	2	2	10	
								00 101 011				
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101	2	2	10	
								00 101 011				

Notas: ss es cualquiera de los pares de registros BC, DE, HL, SP
pp es cualquiera de los pares de registros BC, DE, IX, SP
rr es cualquiera de los pares de registros BC, DE, IY, SP

Notación de los indicadores: • = indicador no está afectado; 0 = indicador a cero;
1 = indicador a 1; X = indicador es desconocido.
↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

forma un poco diferente. Estas instrucciones con su código hex asociado aparecen en la tabla 11-3. Los detalles de como las instrucciones aritméticas de 16 bits manipulan los indicadores de estado aparecen en la tabla 11-5. Como usted puede ver, contiene mucha más información de la que aparece en la tabla 11-3. En el apéndice A, introduciremos un conjunto completo de tablas de instrucciones.

INSTRUCCIONES CP Y DE BUSQUEDA DE BLOQUES: CPI, CPD, CPIR Y CPDR

Las instrucciones *CP* comparan el contenido del acumulador con un byte s de

8 bits especificado, calculando la diferencia A–s y colocando los indicadores de C, Z, P/V, como sobrepasamiento-no paridad, S y H de acuerdo con el resultado. El indicador N se coloca a 1, debido a la resta. Nótese que ni el acumulador ni el byte s cambian como resultado de la operación CP. La diferencia A–s es guardada en alguna parte, interna de la CPU, de forma que el efecto neto de la instrucción CP para el programador es que cambian los indicadores de estado. Así, por ejemplo, la instrucción CP B tiene el mismo efecto en A, B y el registro F que la secuencia

```
LD C,A   ; Guardar el acumulador
SUB B    ; Realiza la resta y coloca los indicadores
LD A,C   ; Restaura el acumulador
```

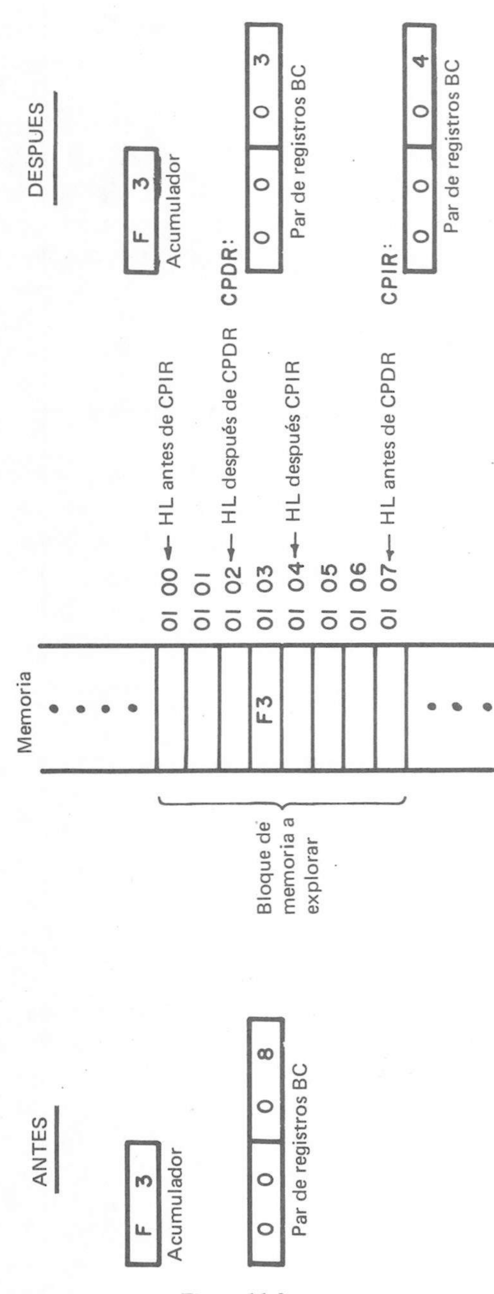
Usted ha visto una aplicación de la instrucción DP en el experimento que demuestra las tablas de salto al final del capítulo 7.

El bloque de instrucciones de búsqueda que aparece en la tabla 11-4 opera en forma similar a las instrucciones de transferencia de bloques. La operación LD es la base para la transferencia de bloque mientras que la instrucción CP es la base para el grupo de búsqueda de bloques. Las *instrucciones de búsqueda de bloque* facilitan el proceso de buscar posiciones de memoria en secuencia hasta encontrar una igualdad con un "byte clave" contenido en el acumulador. Tal y como con las instrucciones de transferencia de bloques, se deben inicializar ciertos registros antes de la ejecución de cualquier instrucción de búsqueda de bloques:

BC = número de posiciones de memoria a explorar
HL = dirección del byte que se va a comparar con el contenido del acumulador
A = el valor clave que se debe encontrar entre bytes sucesivos de memoria

La ejecución de la instrucción CPI, *comparar e incrementar*, hace que ocurran los siguientes pasos:

1. El byte situado en la dirección que señalan el par de registros HL, es comparado con el contenido del acumulador. Los indicadores Z, S y H son colocados de acuerdo con el resultado de la comparación. La instrucción CP (HL) afecta al indicador C, mientras que la instrucción CPI (así como las otras instrucciones de búsqueda) no afectan al indicador C. Así, este paso no es idéntico a la ejecución de una instrucción CP (HL), hablando estrictamente.
2. El contenido del par de registros HL es incrementado.
3. El contenido del par de registros BC es decrementado. En este momento, el indicador de Z se pone a 1 si A = (HL), y a cero en caso contrario. El indicador P/V se pone a cero si el par de registros BC = 0000, y se pone a 1 en caso contrario.



ANTES
F 3
Acumulador
0 0 0 8
Par de registros BC
Memoria
F3
01 00 ← HL antes de CPIR
01 01
01 02 ← HL después de CPDR
01 03
01 04 ← HL después CPIR
01 05
01 06
01 07 ← HL antes de CPDR
Bloque de memoria a explorar
DESPUES
F 3
Acumulador
CPDR:
0 0 0 3
Par de registros BC
CPIR:
0 0 0 4
Par de registros BC

Figura 11-2.

La ejecución de la instrucción CPIR, *comparar-incrementar-repetir,* hace que ocurran las siguientes cosas:

1. El byte que está situado en la posición de memoria direccionado por el par de registros HL es comparado con el contenido del acumulador. Los indicadores Z, S y H se colocan de acuerdo con el resultado.
2. El contenido del par de registros HL es incrementado.
3. El contenido del par de registros BC es decrementado. En este momento el indicador A ha sido colocado si A = (HL), y se pone a cero en caso contrario. El indicador P/V se pone a 0 si el par de registros BC = 0000, y se pone a 1 en caso contrario.
4. Si el par de registros BC = 0000 o bien A = (HL), entonces la ejecución continúa con la próxima instrucción, de lo contrario se repiten los pasos 1, 2 y 3.

La ejecución de las instrucciones CPD, *comparar-decrementar,* y CPDR, *comparar-decrementar-repetir,* resulta en una secuencia de acontecimientos muy similar. La única diferencia es que el Paso 2 decrementa HL. La figura 11-2 ilustra los registros y las posiciones de memoria antes y después de la ejecución de las instrucciones CPIR y CPDR.

INTRODUCCION A LOS EXPERIMENTOS

Los siguientes experimentos están diseñados para darle una idea de como programar el Z-80 para realizar operaciones aritméticas básicas. También se investigan algunas aplicaciones del grupo de instrucciones de búsqueda de bloques.

Los experimentos que usted realizará se pueden resumir de la siguiente forma:

Experimento N.°	Comentarios
1	Demuestra un método para una multiplicación binaria multibyte.
2	Demuestra un método para programar una resta en bcd multibyte.
3	Demuestra un método para programar una operación de división en la cual un número binario de 16 bits es dividido por un número binario de 8 bits para calcular el cociente y el resto.
4	Demuestra las instrucciones de búsqueda de bloques y de comparación, en dos útiles aplicaciones.

EXPERIMENTO N.° 1

Propósito

El propósito de este experimento es el de demostrar un método para programar aritmética binaria de precisión múltiple. El próximo programa listado multiplica dos números binarios de 16 bits almacenados en las posiciones de memoria 0130-0131 y 0132-0133 y coloca el producto en las posiciones 0134-0135. Más adelante le mostraremos un programa que multiplica dos números binarios de igual longitud para producir un producto de la misma longitud. Así, usted podrá ver como su Nanocomputador puede multiplicar números de 64 bits lo mismo que los computadores de gran tamaño. (Hay que decir que solamente la velocidad *no* es comparable.)

Programa N.° 32

Posición de memoria	Código objeto		Código fuente	Comentarios
0100	21 00 00	MLT16:	LD HL,0000H	; HL = producto, inicializarlo a cero
0103	ED 5B 30 01		LD DE,(0130H)	; DE = multiplicando
0107	ED 4B 32 01		LD BC,(0132H)	; BC = multiplicador
010B	7A		LD A,D	; Ver si DE = 0
010C	B3		OR E	
010D	CC 38 00		CALL Z,0038H	; Si DE = 0, saltar al monitor
0110	CB 38	MLT:	SRL B	; BC: desplazamiento a la derecha, colocando cero
0112	CB 19		RR C	; El indicador C es igual al bit
				; que se multiplica
0114	30 04		JR NC,NCF	; Comprobar el indicador C
0116	19		ADD HL,DE	; Indicador C a 1, sumar DE a HL
0117	DC 38 00		CALL C,0038H	; Si ADD provoca arrastre, sobrepasamiento
011A	78	NCF:	LD A,B	; Si el indicador C está a 0, comprobar para ver
011B	B1		OR C	; si BC = 0
011C	CA 29 01		JP Z,ANS	; Si BC = 0 entonces guardar la respuesta
011F	CB 23		SLA E	; De lo contrario desplazar DE hacia la izquierda
0121	CB 12		RL D	
0123	DC 38 00		CALL 0038H	; Si C está a 1, se ha producido sobrepasamiento
				; y hay que volver al monitor
0126	C3 10 01		JP MLT	; De lo contrario continuar
0129	22 34 01	ANS:	LD (0134H),HL	; Guardar la respuesta
012C	FF		RST 38H	; Retorno al monitor

Paso 1

Vamos a discutir la metodología utilizada en este programa para realizar una multiplicación binaria. Para facilitarlo, vamos a multiplicar dos números binarios de 4

bits. El principio que se muestra a continuación para la multiplicación de 4 bits se aplica igualmente a 8 bits, 16 bits o a cualquier otra multiplicación binaria de distinta precisión. Supongamos que deseamos multiplicar 0011 por 0101. Un procedimiento, muy similar a los métodos usuales de la aritmética decimal, es el siguiente:

```
      0 0 1 1     Multiplicando
      0 1 0 1     Multiplicador
    ---------
      0 0 1 1     1 × 0 0 1 1
    0 0 0 0       0 × 0 0 1 1
  0 0 1 1         1 × 0 0 1 1
0 0 0 0           0 × 0 0 1 1
---------------
0 0 0 1 1 1 1
```

Así vemos que se desarrolla un comportamiento claro: Cada vez que hay un bit 1 en el multiplicador se produce una versión desplazada del multiplicando que se sumará a una suma que dará el producto. El desplazamiento, desde luego va colocando ceros desde la derecha. El programa precedente implementa esta técnica de desplazar y sumar tál y como se ha ilustrado previamente. El par de registros HL guarda el producto; el registro BC guarda el multiplicador; y el registro DE guarda el multiplicando.

Paso 2

Cargar y ejecutar el programa de muestra para diferentes pares de números binarios de 16 bits. Obsérvese que mientras que los números se mantienen relativamente pequeños el producto calculado es correcto. Por ejemplo:

0400 × 0020 = 8000
OOFF × OOFF = FE01
0100 × OOFF = FF00

¿Cuánto vale 0100 × 0100 = ?

Obtenemos 0000 en el registro HL lo cual es incorrecto. Vamos a hacer el cálculo a mano. La respuesta es 1 seguido de 16 ceros, o en tres bytes hex,

0 1 0 0 0 0.

Desafortunadamente, solamente se permiten dos bytes en nuestra respuesta, de forma que el byte tres, que es el byte más significativo, no puede aparecer en la respuesta. Este es un caso de *sobrepasamiento* que está detectado en el programa precedente mediante las instrucciones CALL C,0038H y CALL 0038H. En este programa, nosotros solamente insertamos la dirección del sistema operativo en lugar de

la dirección de una rutina que de alguna manera daría una información de la condición de sobrepasamiento al usuario. Nótese que se detecta la condición de sobrepasamiento en dos lugares.

Vamos a observar dos ejemplos de 4 bits para ver porque:

```
   1 0 0 0   Multiplicando
   0 1 1 0   Multiplicador
   -------
   0 0 0 0
 1 0 0 0     Se produce sobrepasamiento debido a que el indicador
             de arrastre se coloca a 1 como resultado de desplazar
             el multiplicando. El indicador de arrastre a 1 señala
             que el producto contiene más de cuatro bits.

               0 1 1 0   Multiplicando
               0 0 1 1   Multiplicador
               -------
               0 1 1 0
             0 1 1 0
Suma parcial 1 0 0 1 0   Existe sobrepasamiento debido a que la operación de
                         desplazar y sumar hace que el indicador de arrastre se
                         coloque a 1, indicando así que el producto contiene
                         más de cuatro bits.
```

Así, para detectar sobrepasamiento en las dos situaciones anteriores, el indicador de arrastre se comprueba después de desplazar el multiplicando (par de registros DE) y sumando después el multiplicando desplazado al par de registros HL para formar una suma parcial.

Paso 3

Las mismas técnicas utilizadas previamente para una multiplicación de 16 bits o de dos bytes puede ser aplicada a la multiplicación de un número binario de n bit, cuando n es un número positivo. La siguiente secuencia de instrucciones multiplica dos números binarios byte-NUM que están guardados en las posiciones de memoria secuenciales NUM con el byte menos significativo empezando en la dirección XNUM y YNUM, respectivamente. El producto es guardado en las posiciones secuenciales de memoria NUM con los bytes menos significativos empezando en la dirección ZNUM.

```
MLTN:   LD B,NUM          ; Cargar el número de bytes por número dentro de
                          ; registro B (NUM = constante hex de un byte)
        LD HL,ZNUM        ; Cargar HL con la dirección del producto ZNUM =
                          ; = constante hex de dos bytes
INIT:   LD (HL),00H       ; Inicializar el producto a ceros en cada byte
        INC HL
        DJNZ INIT
```

```
SHIFTX: LD B,NUM             ; Desplazar el multiplicador XNUM hacia la dere-
                             ; cha de un bit y poner cero: Inicializar el registro
                             ; B al número de bytes
        LD IX,XNUM+NUM-01H   ; IX = dirección del byte más significativo del mul-
                             ; tiplicador (XNUM = constante hex de dos bytes)
        XOR A
RTX:    RR (IX)
        DEC IX
        DJNZ RTX
        JR NC,ZERO           ; Si no hay arrastre, el bit actual en el multiplicador
                             ; es cero, de forma que no hay que sumar el multiplicando
        LD B,NUM             ; Si el arrastre está a 1, el bit actual en el multipli-
                             ; cador es 1 y así sumar el multiplicando al produc-
                             ; to parcial que está almacenado en la posición ZNUM
        LD HL,ZNUM
        LD IY,YNUM           ; YNUM = constante hex de dos bytes
        CCF                  ; Complementar el indicador de arrastre que estaba
                             ; a 1. Así, el indicador C está ahora a cero
NXTBYT: LD A,(IY)            ; Sumar el multiplicando al producto
        ADC (HL)
        LD (HL),A            ; Guardar la suma en la posición de memoria del
                             ; producto
        INC IY               ; Actualizar los indicadores de byte
        INC HL
        DJNZ NXTBYT
ZERO:   LD B,NUM             ; Comprobar si el multiplicador es cero. Si es así,
                             ; hemos terminado
        LD IX,XNUM
        XOR A                ; Borrar A para ver si es cero
ZCHK:   OR (IX)              ; Comprobar si el multiplicador = 0
        JR NZ,SHIFTY         ; Si el multiplicador no es cero, entonces desplazar
                             ; de nuevo el multiplicando
        INC IX
        DJNZ ZCHK
        RST 38H              ; Volver al monitor si el multiplicador es cero
SHIFTY: LD B,NUM             ; Desplazar el multiplicando YNUM hacia la dere-
                             ; cha de un byte, colocando a cero el bit vacío
        LD IY,YNUM
        XOR A                ; Borrar el bit de arrastre
LFTY:   RL (IY)              ; Empezar el desplazamiento
        INC IY
        DJNZ LFTY
        JP SHIFTX
        RST 38H              ; Multiplicador = 0, así, se ha terminado.
```

Repase esta rutina cuidadosamente para asegurarse de que la entiende completamente. Por primera vez hemos utilizado variables como NUM, XNUM, YNUM y ZNUM para constantes hex de un byte y de dos bytes. Esta es una práctica muy utilizada en la literatura sobre el desarrollo del software, pero esto usted debe acostumbrarse a esta técnica.

Paso 4

Ensamble a mano el programa de multiplicación NUM-byte substituyendo sus propios valores para NUM, XNUM, YNUM y ZNUM. Vea si usted puede ejecutar algunos programas de prueba.

Paso 5

Observe que el programa de multiplicación NUM-byte no comprueba el sobrepasamiento. Coloque sus propias comprobaciones y compruebe si funciona.

EXPERIMENTO N.° 2

Propósito

El propósito de este experimento es el de demostrar un método para programar la resta bcd multibyte. El siguiente programa entra dos números bcd NUM-byte cuyos bytes menos significativos están en las posiciones XNUM y YNUM, respectivamente, calcula su resta (el número empieza en la dirección XNUM), y guarda la resta empezando en la dirección ZNUM.

Programa N.° 33

Posición de memoria	Código objeto	Código fuente	Comentarios
0200	06 N	SUBN: LD B,NUM	; El registro B cuenta el ; número de bytes
0202	DD 21 X2 X1	LD IX,XNUM	
0206	FD 21 Y2 Y1	LD IY,YNUM	
020A	21 Z2 Z1	LD HL,ZNUM	
020D	37	SCF	; Colocar el indicador de arrastre: ; complemento a 100 para la primera resta
020E	3E 99	NXTBYT: LD A,99H	; Encontrar el complemento a 99 o 100 ; del substraendo
0210	CE 00	ADC A,00H	
0212	FD 96 00	SUB (IY)	
0215	DD 86 00	ADD A,(IX)	; Sumar el byte del minuendo
0218	27	DAA	; Ajuste para aritmética ; decimal
0219	77	LD (HL),A	
021A	DD 23	INC IX	; Actualizar indicadores
021C	FD 23	INC IY	
021E	23	INC HL	

```
021F    10 E8    DJNZ NXTBYT   ; Continuar restando hasta que se
                               ; hayan procesado todos los bytes
                               ; Si no hay arrastre después del último
                               ; byte ¡es que hay sobrepasamiento!
0221    FF       RST 38H
```

Paso 1

Vamos a discutir como trabaja este programa. Recuerde que en nuestra discusión de la resta de números en complemento a dos, dijimos que restar un número en complemento a dos es equivalente a calcular su complemento a dos y entonces sumarlo. Esto mismo es cierto para la resta bcd empaquetada, solamente que en lugar de formar el complemento a dos, usted forma el complemento a 100. Vamos a mirar algunos ejemplos:

Byte BCD empaquetado	03	94	30	01	50
Complemento a 100	97	06	70	99	50

Así, para encontrar el complemento a 100 de un byte bcd empaquetado, simplemente reste el byte desde 100 como un número decimal de dos dígitos. Similarmente, usted puede calcular el complemento a 10 de un nibble bcd, pero no necesitamos utilizar esto aquí. Vamos ahora a realizar una resta bcd de tres byte utilizando la técnica de *complementar y sumar.*

Encontrar la resta entre: 256925–133639

Paso 1: Formar el complemento a 100 del byte menos significativo del substraendo

$$100 - 39 = 61$$

Paso 2: Sumar 61 al byte menos significativo del minuendo (suma decimal)

$$25 + 61 = 86$$

No se ha producido arrastre.

Paso 3: Puesto que en el Paso 2 no se produjo arrastre, forme el *complemento a 99* del byte menos significativo del substraendo

$$99 - 36 = 63$$

Paso 4: Sumar 63 al próximo byte menos significativo del minuendo

$$69 + 63 = 32$$

Se ha producido arrastre

Paso 5: Puesto que en el Paso 4 se ha producido un arrastre, formar el complemento a 100 del byte más significativo del sustraendo

$$100 - 13 = 87$$

Paso 6: Sumar 87 al byte más significativo del minuendo

$$25 + 87 = 12$$

Se ha producido arrastre

Paso 7: La respuesta es 123286 que es correcta. Decimos que la respuesta es correcta porque se ha producido arrastre. Si no hubiera habido arrastre la respuesta habría sido incorrecta. Puesto que los números bcd son siempre mayores o iguales a cero, se produce sobrepasamiento siempre que el sustraendo es mayor que el minuendo.

No profundizaremos en la prueba matemática de la técnica precedente. Es suficiente decir que el complemento a 100 se forma cuando no se ha producido un arrastre en la resta desde el próximo byte de más peso, mientras que se forma el complemento a 99, cuando se ha producido un arrastre al realizar la resta de un número bcd empaquetado de otro. En términos de la técnica complementar y sumar:

Un arrastre en el proceso de resta es equivalente a que no se produzca arrastre en el proceso de complemento y suma.

Así, si el indicador de arrastre está a 1 al final de la resta bcd, *no* existe sobrepasamiento. Esto puede parecer al principio "poco intuitivo", pero será más natural a medida que usted piense en ello.

Paso 2

Cargue y ejecute el programa anterior con varios ejemplos de grupos de caracteres bcd para restar. Asegúrese de proporcionar valores para:

NUM = byte constante hex que representa el número de bytes bcd empaquetados
XNUM = 2 byte de dirección del minuendo (hex)
YNUM = 2 byte dirección del substraendo (hex)
ZNUM = 2 byte dirección de la resta (hex)

EXPERIMENTO N.° 3

El propósito de este experimento es el de demostrar un método para programar una operación de división en la cual un número binario de 16 bits es dividido por

un número binario de 8 bits para calcular un cociente y un resto. En el programa que se lista a continuación, se supone que inicialmente los registros HL contienen el dividendo binario de 16 bits y el registro D contiene el divisor binario de 8 bits. Al finalizar la ejecución, el cociente de 8 bits está en el registro L y el resto de 8 bits está en el registro H. Para que el algoritmo de división implementado en el programa 34 funcione correctamente, suponemos que el divisor y el dividendo están en *forma normalizada.* Esto es:

a. El bit más significativo del dividendo de 16 bits es cero, y
b. El bit más significativo del dividendo es menor que el divisor para asegurar de que el cociente cabrá dentro de los 8 bits destinados a él.

Programa N.° 34

Posición de memoria	Código objeto		Código fuente	Comentarios
0300	06 08	DIV:	LD B,08H	; # bits en el divisor
0302	1E 00		LD E,00H	; Divisor en DE
0304	29	NXTBIT:	ADD HL,HL	; Desplazar HL a la izquierda, colocar a cero
0305	AF		XOR A	; Poner a cero el indicador de arrastre
0306	ED 52		SBC HL,DE	; ¿Cabrán DE?
0308	23		INC HL	; Supongamos que sí
0309	30 02		JR NC,NXT	; Si no, arreglar lo hecho
030B	19		ADD HL,DE	; Sumar DE de nuevo
030C	2B		DEC HL	; Colocar el bit del cociente a 0
030D	10 F5	NXT:	DJNZ NXTBIT	
030F	FF		RST 38H	

Paso 1

Vamos primero a mirar como funciona este programa. El algoritmo utilizado es muy similar al método utilizado para calcular a mano largos problemas de división. Sin embargo es más fácil debido a que solamente existen ceros y unos. Vamos a observar un ejemplo en el cual el número binario de 4 bits N.° 8 es dividido por el número binario de 8 bits 6E. Plantee el problema como lo haría para un problema de división larga, escribiendo los números en binario:

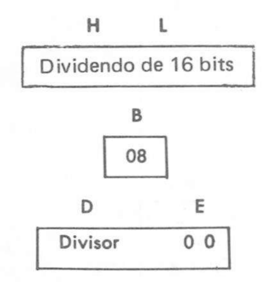
$$1\,0\,0\,0\,\overline{)\,0\,1\,1\,0\,1\,1\,1\,0}$$

Para determinar los bits sucesivos del cociente, simplemente entre un 1 si el divisor "entrará" o un 0 si "no entra" en los bits más significativos del resto de restas sucesivas del divisor desde el dividendo:

```
                 1 1 0 1
1 0 0 0 | 0 1 1 0 1 1 1 0
            1 0 0 0
            -------
            1 0 1 1
            1 0 0 0
            -------
              0 1 1 1 0
                1 0 0 0
                -------
                  1 1 0
```

Así el cociente es 1101 = D (base 16) y el resto es 0110 = 6 (base 16).

En el programa precedente los registros son inicializados de la siguiente forma:



Puesto que el bit más significativo de HL es cero y el contenido de D es mayor que el contenido de H (ver las definiciones acerca de la normalización del dividendo y del divisor), el primer paso en el programa es efectuar un desplazamiento hacia la izquierda, colocar a cero HL de un bit (ADD HL,HL) y entonces comparar el contenido de DE y HL. Si DE es menor que o igual a HL, el cociente se coloca a 1, de lo contrario es cero. Obsérvese que el cociente es desplazado dentro de los cuatro bits de menor peso del par de registros HL. Esto es posible porque a medida de que cada bit del cociente es sumado, se descarta el viejo bit del dividendo. El método para determinar cuando el bit del nuevo cociente (que siempre es el bit menos significativo del registro L) debe ser cero a uno, DE se resta de HL. Un arrastre indica que el bit del cociente debe ser cero, mientras que si no hay arrastre implica que DE "cabe dentro" de HL de forma que el bit del cociente debe ser 1 (INC HL). Nótese que la resta HL–DE debe utilizar la instrucción SBC puesto que no existe una instrucción SUB de 16 bits. Esto necesita la instrucción XOR A precisamente antes de la instrucción SBC para asegurarse de que el indicador de arrastre sea cero. Si DE "no caben" dentro de HL, el valor del último resto en HL debe ser restaurado añadiendo de nuevo DE y decrementando HL.

Paso 2

Cargue y ejecute el programa precedente en modo paso a paso para varios programas de muestra.

EXPERIMENTO N.º 4

Propósito

El propósito de este experimento es el de demostrar las instrucciones de búsqueda de bloque y de comparación en dos aplicaciones útiles. Presentamos dos programas que utilizan estas instrucciones para realizar tareas de programación frecuentemente utilizadas.

Programa N.º 35

BUSCAR en una serie de caracteres un determinado carácter.

Posición de memoria	Código objeto	Código fuente	Comentarios
0400	21 00 0A	LD HL,0A00H	; Dirección inicial de la ; serie de caracteres
0403	01 20 00	LD BC,0020H	; Número de caracteres ; a comparar
0406	3E 24	LD A,24H	; Caracter a localizar: ; 24 es un '$' ASCII
0408	ED B1	CPIR	; Encontrar el $
040A	C2 0F 04	JP NZ,NOFIND	; Si el indicador Z = 0 no ; se ha encontrado el carácter
040D	2B	DEC HL	; Restar 1 de HL de forma que ; señalen al carácter
040E	03	INC BC	; Incrementar BC para que dé el
040F	FF	NOFIND: RST 38H	; número del carácter en la serie ; de caracteres

Programa N.º 36

Buscar en una tabla de siglas una determinada sigla identificada por una serie de tres caracteres.

Posición de memoria	Código objeto	Código fuente	Comentarios
0412	31 00 0F	LD SP,0F00H	; Situar el stack
0415	21 00 0C	LD HL,0C00H	; Dirección del último grupo en la tabla
0418	01 06 00	LD BC,0006H	; Número de grupos en la tabla

```
041B    3A 00 0B            LD A,(0B00H)     ; Primer carácter para identificar el
                                             ; grupo de 3 caracteres
041E    11 F9 FF            LD DE,FFF9H      ; —(longitud del grupo —1): 16 bits
                                             ; en complemento a dos
0421    ED A9      NREC:    CPD              ; ¿Igualdad?
0423    28 09               JR Z,CHECK
0425    E2 00 1A            JP PO,NOFIND     ; ¿Se han examinado todos los grupos?
0428    19         UPD:     ADD HL,DE        ; Actualizar HL al inicio del próximo
                                             ; grupo
0429    3A 00 0B            LD A,(0B00H)     ; Inicializar el acumulador
042C    18 F3               JR NREC          ; Mirar al próximo grupo
042E    3A 01 0B   CHECK:   LD A,(0B01H)     ; Comparar el segundo byte
0431    E5                  PUSH HL          ; HL señala ahora al último byte del
                                             ; grupo inmediatamente precedente
                                             ; para ser comparado
0432    DD E1               POP IX           ; Cargar IX con HL
0434    DD BE 02            CP (IX+02H)
0437    20 EF               JR NZ,UPD        ; Ir atrás a buscar si no hay igualdad
0439    3A 02 0B            LD A,(0B02H)     ; Comparar el tercer byte
043C    DD BE 03            CP (IX+03H)
043F    20 E7               JR NZ,UPD        ; Ir atrás a buscar si no hay igualdad
0441    23                  INC HL           ; HL señala ahora al primer byte del
                                             ; grupo que coincide; el indicador
                                             de cero = 1
0442    FF         NOFIND:  RST 38H
```

Paso 1

Vamos a examinar el Programa N.° 35. Este programa empieza por cargar el par de registros HL con la dirección del byte 1 de un grupo de bytes, el par de registros BC con el número de bytes en este conjunto y el acumulador con el byte clave, es decir el byte que se quiere enconrrar en esta búsqueda. La instrucción CPIR comprueba secuencialmente cada byte en el conjunto hasta que se encuentra una igualdad o hasta que no hay más bytes a comprobar. Si se encuentra una igualdad, la instrucción CPIR coloca a 1 el indicador de cero (Z) lo cual hace que se ejecuten las instrucciones DEC HL y INC BC de forma que HL siga señalando al conjunto de bytes a comparar, y BC representa el número de byte en el conjunto de bytes. Si no se encuentra una igualdad, se devuelve el control al sistema operativo con el indicador de cero colocado a 1, el indicador P/V = 0, y BC = 00.

Este programa si se diseña como una subrutina puede ser utilizado para implementar tablas de salto como la que se ha demostrado en el capítulo 7. Los valores a buscar se pueden guardar en un bloque separado de memoria lejos de sus direcciones de salto asociadas. Se llamará al programa N.° 35 para buscar los valores para una igualdad y retornará el índice del valor encontrado en el par de registros BC. Este índice se utilizará entonces para encontrar las direcciones de salto adecuadas para la subsiguiente transferencia de control.

Paso 2

Cargue y ejecute el programa N.º 35. Utilice varios conjuntos de prueba y distintos valores clave para comprobar completamente la lógica del programa.

Paso 3

Vamos a examinar el programa N.º 36. Supongamos que tenemos una tabla situada en la memoria de la siguiente forma:

	Byte N.º								
	1	2	3	4	5	6	7	8	
Posición									
0BD8	41	51	46	31	32	33	30	36	Grupo 1
0BE0	42	46	47	33	36	30	30	34	Grupo 2
0BE8	41	42	43	36	35	34	32	31	Grupo 3
0BF0	43	42	41	36	36	36	36	36	Grupo 4
0BF8	43	41	42	34	33	34	33	34	Grupo 5
0C00	42	42	42	31	32	33	32	31	Grupo 6
	Iniciales de identificación			Extensión telefónica					

La tabla consiste en seis filas y cada fila contiene 8 bytes. Para cada fila:

Los bytes 1-3 son un código de identificación de tres caracteres
Los bytes 4-8 representan una extensión de teléfono de 5 dígitos

Así, tenemos un directorio telefónico para una oficina de seis personas. Para mirar el número de teléfono de una persona, simplemente haga coincidir la representación ASCII de sus iniciales con los tres primeros bytes de alguna fila en la tabla, y los próximos cinco bytes son el número de teléfono. Aquí está una tabla de equivalencias ASCII-HEX con las letras del alfabeto.

A	41	J	4A	S	53
B	42	K	4B	T	54
C	43	L	4C	U	55
D	44	M	4D	V	56
E	45	N	4E	W	57
F	46	O	4F	X	58
G	47	P	50	Y	59
H	48	Q	51	Z	5A
I	49	R	52		

Recuerde de un experimento anterior que la representación ASCII para los caracteres 0 al 9 va desde 30 al 39. Así "AQF" tiene el número de teléfono 12306.

El programa N.° 36 lee una tabla tal como la que se ha presentado anteriormente y devuelve un indicador al conjunto cuyos tres primeros caracteres coinciden con el conjunto clave, guardado en las posiciones de memoria 0B00, 0B01 y 0B02. El programa empieza cargando el par de registros HL con la dirección del último grupo, el par de registros BC con el número de grupos de la tabla y el acumulador con el primer byte del grupo clave. La instrucción CPD en combinación de sumas sucesivas de –7 al par de registros HL comprueba los primeros bytes de cada grupo, desde el grupo número 6 al grupo número 35, para encontrar una igualdad con el byte 1 del grupo clave de tres bytes. (Obsérvese que se suma –7 a HL en lugar de –8 porque la instrucción CPD decrementa HL.) Cuando se encuentra una igualdad, se deben comprobar los bytes dos y tres. Esto se realiza mediante la secuencia de instrucciones que empieza en CHECK. Esta secuencia se puede cambiar fácilmente para adaptarla a la necesidad de comprobar mayor número de bytes introduciendo un bucle, pero solamente para dos bytes, no es necesario un bucle.

APENDICE **A**

Resumen de los códigos de operación y de los tiempos de ejecución del Z-80

Las tablas siguientes resumen el conjunto de instrucciones del Z-80. Las instrucciones están arregladas en grupos como se mostró anteriormente en las tablas de instrucciones que aparecieron en los capítulos del 6 al 12. Cada tabla muestra el mnemónico en lenguaje ensamblador, una descripción simbólica abreviada del funcionamiento de la instrucción, el código de operación binario, el número de bytes, así como el número de ciclos de memoria y el número de estados T (períodos externos de reloj) necesarios para la lectura y ejecución de la instrucción. Cuando es necesario, se incluyen comentarios adicionales.

Tabla A-1. El grupo de carga de 8 bits

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de ciclos T	Comentarios
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	•	•	•	•	00 r 110	2	2	7	000 B
								← n →				001 C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01 r 110	1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11 011 101	3	5	19	011 E
								01 r 110				100 H
								← d →				101 L
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11 111 101	3	5	19	111 A
								01 r 110				
								← d →				
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01 110 r	1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11 011 101	3	5	19	
								01 110 r				
								← d →				
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11 111 101	3	5	19	
								01 110 r				
								← d →				
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00 110 110	2	3	10	
								← n →				
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11 011 101	4	5	19	
								00 110 110				
								← d →				
								← n →				
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11 111 101	4	5	19	
								00 110 110				
								← d →				
								← n →				
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00 001 010	1	2	7	
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00 011 010	1	2	7	
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00 111 010	3	4	13	
								← n →				
								← n →				
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00 000 010	1	2	7	
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00 010 010	1	2	7	
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00 110 010	3	4	13	
								← n →				
								← n →				
LD A, I	A ← I	•	↕	IFF	↕	0	0	11 101 101	2	2	9	
								01 010 111				
LD A, R	A ← R	•	↕	IFF	↕	0	0	11 101 101	2	2	9	
								01 011 111				
LD I, A	I ← A	•	•	•	•	•	•	11 101 101	2	2	9	
								01 000 111				
LD R, A	R ← A	•	•	•	•	•	•	11 101 101	2	2	9	
								01 001 111				

Notas: r, r' significa cualquiera de los registros A, B, C, D, E, H, L

IFF el contenido de la báscula de habilitación de las interrupciones (IFF) es copiado en el indicador P/V

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-2. El grupo de carga de 16 bits

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
LD dd, nn	dd ← nn	●	●	●	●	●	●	00 dd0 001	3	3	10	dd Par
								← n →				00 BC
								← n →				01 DE
LD IX, nn	IX ← nn	●	●	●	●	●	●	11 011 101	4	4	14	10 HL
								00 100 001				11 SP
								← n →				
								← n →				
LD IY, nn	IY ← nn	●	●	●	●	●	●	11 111 101	4	4	14	
								00 100 001				
								← n →				
								← n →				
LD HL, (nn)	H ← (nn+1)	●	●	●	●	●	●	00 101 010	3	5	16	
	L ← (nn)							← n →				
								← n →				
LD dd, (nn)	dd_H ← (nn+1)	●	●	●	●	●	●	11 101 101	4	6	20	
	dd_L ← (nn)							01 dd1 011				
								← n →				
								← n →				
LD IX, (nn)	IX_H ← (nn+1)	●	●	●	●	●	●	11 011 101	4	6	20	
	IX_L ← (nn)							00 101 010				
								← n →				
								← n →				
LD IY, (nn)	IY_H ← (nn+1)	●	●	●	●	●	●	11 111 101	4	6	20	
	IY_L ← (nn)							00 101 010				
								← n →				
								← n →				
LD (nn), HL	(nn+1) ← H	●	●	●	●	●	●	00 100 010	3	5	16	
	(nn) ← L							← n →				
								← n →				
LD (nn), dd	(nn+1) ← dd_H	●	●	●	●	●	●	11 101 101	4	6	20	
	(nn) ← dd_L							01 dd0 011				
								← n →				
								← n →				
LD (nn), IX	(nn+1) ← IX_H	●	●	●	●	●	●	11 011 101	4	6	20	
	(nn) ← IX_L							00 100 010				
								← n →				
								← n →				
LD (nn), IY	(nn+1) ← IY_H	●	●	●	●	●	●	11 111 101	4	6	20	
	(nn) ← IY_L							00 100 010				
								← n →				
								← n →				
LD SP, HL	SP ← HL	●	●	●	●	●	●	11 111 001	1	1	6	
LD SP, IX	SP ← IX	●	●	●	●	●	●	11 011 101	2	2	10	
								11 111 001				
LD SP, IY	SP ← IY	●	●	●	●	●	●	11 111 101	2	2	10	
								11 111 001				qq Par
PUSH qq	(SP-2) ← qq_L	●	●	●	●	●	●	11 qq0 101	1	3	11	00 BC
	(SP-1) ← qq_H											01 DE
PUSH IX	(SP-2) ← IX_L	●	●	●	●	●	●	11 011 101	2	4	15	10 HL
	(SP-1) ← IX_H							11 100 101				11 AF
PUSH IY	(SP-2) ← IY_L	●	●	●	●	●	●	11 111 101	2	4	15	
	(SP-1) ← IY_H							11 100 101				
POP qq	qq_H ← (SP+1)	●	●	●	●	●	●	11 qq0 001	1	3	10	
	qq_L ← (SP)											
POP IX	IX_H ← (SP+1)	●	●	●	●	●	●	11 011 101	2	4	14	
	IX_L ← (SP)							11 100 001				
POP IY	IY_H ← (SP+1)	●	●	●	●	●	●	11 111 101	2	4	14	
	IY_L ← (SP)							11 100 001				

Notas: dd es cualquiera de los pares de registros BC, DE, HL, SP
qq es cualquiera de los pares de registros AF, BC, DE, HL
$(PAR)_H$, $(PAR)_L$ se refiere a los ocho bits de mayor y menor peso respectivamente, del par de registros. Ej. $BC_L = C$, $AF_H = A$

Notación de los indicadores: ● = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-3. Grupo de intercambio y transferencia y búsqueda de bloques

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	(BC, DE, HL) ↔ (BC', DE', HL')	•	•	•	•	•	•	11 011 001	1	1	4	Intercambio entre el grupo de registros y el grupo de registros auxiliares
EX (SP), HL	H ↔ (SP+1) L ↔ (SP)	•	•	•	•	•	•	11 100 011	1	5	19	
EX (SP), IX	IX_H ↔ (SP+1) IX_L ↔ (SP)	•	•	•	•	•	•	11 011 101 11 100 011	2	6	23	
EX (SP), IY	IY_H ↔ (SP+1) IY_L ↔ (SP)	•	•	•	•	•	•	11 111 101 11 100 011	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	① ↕	•	0	0	11 101 101 10 100 000	2	4	16	Cargar (HL) en (DE), incrementar los indicadores y decrementar el contador de byte (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repetir hasta BC = 0	•	•	0	•	0	0	11 101 101 10 110 000	2 2	5 4	21 16	Si BC ≠ 0 Si BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	① ↕	•	0	0	11 101 101 10 101 000	2	4	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repetir hasta BC = 0	•	•	0	•	0	0	11 101 101 10 111 000	2 2	5 4	21 16	Si BC ≠ 0 Si BC = 0
CPI	A - (HL) HL ← HL+1 BC ← BC-1	•	② ↕	① ↕	↕	1	↕	11 101 101 10 100 001	2	4	16	
CPIR	A - (HL) HL ← HL+1 BC ← BC-1 Repetir hasta A = (HL) o BC = 0	•	② ↕	① ↕	↕	1	↕	11 101 101 10 110 001	2 2	5 4	21 16	Si BC ≠ 0 y A ≠ (HL) Si BC = 0 o A = (HL)
CPD	A - (HL) HL ← HL-1 BC ← BC-1	•	② ↕	① ↕	↕	1	↕	11 101 101 10 101 001	2	4	16	
CPDR	A - (HL) HL ← HL-1 BC ← BC-1 Repetir hasta A = (HL) o BC = 0	•	② ↕	① ↕	↕	1	↕	11 101 101 10 111 001	2 2	5 4	21 16	Si BC ≠ 0 y A ≠ (HL) Si BC = 0 o A = (HL)

Notas:
1 El indicador P/V es 0 si el resultado de BC − 1 = 0, de lo contrario P/V = 1
2 El indicador Z es 1 si A = (HL), de lo contrario Z = 0.

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-4. El grupo aritmético y lógico de 8 bits

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
ADD r	A ← A + r	↕	↕	V	↕	0	↕	10 000 r	1	1	4	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD n	A ← A + n	↕	↕	V	↕	0	↕	11 000 110 ← n →	2	2	7	
ADD (HL)	A ← A + (HL)	↕	↕	V	↕	0	↕	10 000 110	1	2	7	
ADD (IX+d)	A←A + (IX+d)	↕	↕	V	↕	0	↕	11 011 101 10 000 110 ← d →	3	5	19	
ADD (IY+d)	A←A+(IY+d)	↕	↕	V	↕	0	↕	11 111 101 10 000 110 ← d →	3	5	19	
ADC s	A ← A + s + CY	↕	↕	V	↕	0	↕	001				s es cualquiera de r, n, (HL), (IX + d), (IY + d) como se muestra en la instrucción ADD
SUB s	A ← A − s	↕	↕	V	↕	1	↕	010				
SBC s	A ← A − s − CY	↕	↕	V	↕	1	↕	011				
AND s	A ← A ∧ s	0	↕	P	↕	0	↕	100				
OR s	A ← A ∨ s	0	↕	P	↕	0	↕	110				Los bits indicados reemplazan el 000 con el grupo ADD anterior
XOR s	A ← A • s	0	↕	P	↕	0	↕	101				
CP s	A − s	↕	↕	V	↕	1	↕	111				
INC r	r ← r + 1	•	↕	V	↕	0	↕	00 r 100	1	1	4	
INC (HL)	(HL) ← (HL)+1	•	↕	V	↕	0	↕	00 110 100	1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	•	↕	V	↕	0	↕	11 011 101 00 110 100 ← d →	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d) + 1	•	↕	V	↕	0	↕	11 111 101 00 110 100 ← d →	3	6	23	
DEC d	d ← d − 1	•	↕	V	↕	1	↕	101				d es cualquiera de r, (HL), (IX + d), (IY + d) como se muestra para INC. Mismo formato y estados que para INC. Reemplazar 100 con 101 en el código de operación.

Notas: El símbolo V en la columna del indicador P/V señala que el indicador P/V contiene el sobrepasamiento del resultado de la operación. Similarmente el símbolo P indica paridad. V = 1 significa sobrepasamiento, V = 0 indica que no hay sobrepasamiento, P = 1 significa paridad si el resultado es par, P = 0 significa paridad si el resultado es impar.

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-5. Grupo aritmético y de control de la CPU de aplicación general

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
DAA	Convierte el contenido del Ac. a BCD empaquetado siguiendo a la suma o resta con operandos BCD empaquetados	↕	↕	P	↕	•	↕	00 100 111	1	1	4	Ajuste decimal del acumulador
CPL	$A \leftarrow \overline{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complementa el acumulador (complemento a uno)
NEG	$A \leftarrow 0 - A$	↕	↕	V	↕	1	↕	11 101 101 01 000 100	2	2	8	Cambia el signo al acumulador (complemento a dos)
CCF	$CY \leftarrow \overline{CY}$	↕	•	•	•	0	X	00 111 111	1	1	4	Complementa el indicador de arrastre
SCF	$CY \leftarrow 1$	1	•	•	•	0	0	00 110 111	1	1	4	Pone a 1 el indicador de arrastre
NOP	No operación	•	•	•	•	•	•	00 000 000	1	1	4	
HALT	CPU parada	•	•	•	•	•	•	01 110 110	1	1	4	
DI	$IFF \leftarrow 0$	•	•	•	•	•	•	11 110 011	1	1	4	
EI	$IFF \leftarrow 1$	•	•	•	•	•	•	11 111 011	1	1	4	
IM 0	Colocar el modo 0 de interrupción	•	•	•	•	•	•	11 101 101 01 000 110	2	2	8	
IM 1	Colocar el modo 1 de interrupción	•	•	•	•	•	•	11 101 101 01 010 110	2	2	8	
IM2	Colocar el modo 2 de interrupción	•	•	•	•	•	•	11 101 101 01 011 110	2	2	8	

Notas: IFF indica la bascula de habilitación de interrupciones
CY indica la báscula de arrastre.

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-6. El grupo aritmético de 16 bits

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios	
ADD HL, ss	HL ← HL+ss	↕	•	•	•	0	X	00 ss1 001	1	3	11	ss	Reg.
												00	BC
ADC HL, ss	HL←HL+ ss +CY	↕	↕	V	↕	0	X	11 101 101	2	4	15	01	DE
								01 ss1 010				10	HL
												11	SP
SBC HL, ss	HL←HL- ss -CY	↕	↕	V	↕	1	X	11 101 101	2	4	15		
								01 ss0 010					
ADD IX, pp	IX ← IX + pp	↕	•	•	•	0	X	11 011 101	2	4	15	pp	Reg.
								00 pp1 001				00	BC
												01	DE
												10	IX
												11	SP
ADD IY, rr	IY←IY+ rr	↕	•	•	•	0	X	11 111 101	2	4	15	rr	Reg.
								00 rr1 001				00	BC
												01	DE
												10	IY
												11	SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6		
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101	2	2	10		
								00 100 011					
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101	2	2	10		
								00 100 011					
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6		
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101	2	2	10		
								00 101 011					
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101	2	2	10		
								00 101 011					

Notas: ss es cualquiera de los pares de registros BC, DE, HL, SP
pp es cualquiera de los pares de registros BC, DE, IX, SP
rr es cualquiera de los pares de registros BC, DE, IY, SP

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-7. Grupo de rotación y desplazamiento

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
RLCA	CY ← 7 ← 0 ; A	↕	•	•	•	0	0	00 000 111	1	1	4	Rotación hacia la izquierda circular del acumulador
RLA	CY ← 7 ← 0 ; A	↕	•	•	•	0	0	00 010 111	1	1	4	Rotación izquierda del acumulador
RRCA	7 → 0 → CY ; A	↕	•	•	•	0	0	00 001 111	1	1	4	Rotación circular derecha del acumulador
RRA	7 → 0 → CY ; A	↕	•	•	•	0	0	00 011 111	1	1	4	Rotación a la derecha del acumulador
RLC r	CY ← 7 ← 0 ; r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	11 001 011 00 000 r	2	2	8	Rotación izquierda circular del registro r
RLC (HL)		↕	↕	P	↕	0	0	11 001 011 00 000 110	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		↕	↕	P	↕	0	0	11 011 101 11 001 011 ← d → 00 000 110	4	6	23	
RLC (IY+d)		↕	↕	P	↕	0	0	11 111 101 11 001 011 ← d → 00 000 110	4	6	23	
RL s	CY ← 7 ← 0 ; s = r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	010				El formato de la instrucción y estados es como se muestra para RLC,s. Para formar un nuevo código de operación reemplazar 000 por RLC,s con el código mostrado
RRC s	7 → 0 → CY ; s = r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	001				
RR s	7 → 0 → CY ; s = r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	011				
SLA s	CY ← 7 ← 0 ← 0 ; s = r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	100				
SRA s	7 → 0 → CY ; s = r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	101				
SRL s	0 → 7 → 0 → CY ; s = r, (HL), (IX+d), (IY+d)	↕	↕	P	↕	0	0	111				
RLD	A 7 4 3 0 ; 7 4 3 0 (HL)	•	↕	P	↕	0	0	11 101 101 01 101 111	2	5	18	Rotación de dígito a la izquierda y derecha entre el acumulador y la posición (HL). El contenido de la mitad alta del acumulador no queda afectado
RRD	A 7 4 3 0 ; 7 4 3 0 (HL)	•	↕	P	↕	0	0	11 101 101 01 100 111	2	5	18	

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-8. Grupo bit SET, RESET y TEST

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
BIT b, r	$Z \leftarrow \overline{r_b}$	•	↕	X	X	0	1	11 001 011	2	2	8	
								01 b r				
BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	•	↕	X	X	0	1	11 001 011	2	3	12	
								01 b 110				
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)}_b$	•	↕	X	X	0	1	11 011 101	4	5	20	
								11 001 011				
								← d →				
								01 b 110				
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)}_b$	•	↕	X	X	0	1	11 111 101	4	5	20	
								11 001 011				
								← d →				
								01 b 110				
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011	2	2	8	
								[11] b r				
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011	2	4	15	
								[11] b 110				
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101	4	6	23	
								11 001 011				
								← d →				
								[11] b 110				
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101	4	6	23	
								11 001 011				
								← d →				
								[11] b 110				
RES b, s	$s_b \leftarrow 0$ $s \equiv r$, (HL), (IX+d), (IY+d)							[10]				Para formar un nuevo código de operación reemplazar [11] de SET b,s con [10]. Los indicadores y estados T para la instrucción SET

r	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	Bit comprobado
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notas: La notación s_b indica bit (0 a 7) o posición s.

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-9. Grupo de JUMP (salto)

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011	3	3	10	
								← n →				
								← n →				cc Condición
JP cc, nn	Si la condición cc es cierta PC ← nn, de lo contrario continuar	•	•	•	•	•	•	11 cc 010	3	3	10	000 NZ no cero
								← n →				001 Z cero
								← n →				010 NC no arrastre
												011 C arrastre
												100 PO paridad impar
												101 PE paridad par
												110 P signo positivo
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000	2	3	12	111 M signo negativo
								← e-2 →				
JR C, e	Si C = 0 continuar	•	•	•	•	•	•	00 111 000	2	2	7	Si no se ha alcanzado la condición
								← e-2 →				
	Si C = 1 PC ← PC + e								2	3	12	Si se ha alcanzado la condición
JR NC, e	Si C = 1 continuar	•	•	•	•	•	•	00 110 000	2	2	7	Si no se ha alcanzado la condición
								← e-2 →				
	Si C = 0 PC ← PC + e								2	3	12	Si se ha alcanzado la condición
JR Z, e	Si Z = 0 continuar	•	•	•	•	•	•	00 101 000	2	2	7	Si no se ha alcanzado la condición
								← e-2 →				
	Si Z = 1 PC ← PC + e								2	3	12	Si se ha alcanzado la condición
JR NZ, e	Si Z = 1 continuar	•	•	•	•	•	•	00 100 000	2	2	7	Si no se ha alcanzado la condición
								← e-2 →				
	Si Z = 0 PC ← PC + e								2	3	12	Si se ha alcanzado la condición
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	1	1	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101	2	2	8	
								11 101 001				
JP (IY)	PC ← IY	•	•	•	•	•	•	11 111 101	2	2	8	
								11 101 001				
DJNZ,e	B ← B-1 Si B = 0 continuar	•	•	•	•	•	•	00 010 000	2	2	8	Si B = 0
								← e-2 →				
	Si B ≠ 0 PC ← PC + e								2	3	13	Si B ≠ 0

Notas: e representa la extensión en el modo de direccionamiento relativo.

e es un número con signo en complemento a dos en la gama $<-126, 129>$

e−2 en el código de operación proporciona una dirección efectiva de pc + e mientras PC es incrementado de 2 antes de la suma de e.

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-10. Grupo CALL y RETURN

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
CALL nn	(SP-1)←PC_H (SP-2)←PC_L PC←nn	•	•	•	•	•	•	11 001 101 ← n → ← n →	3	5	17	
CALL cc, nn	Si la condición cc es falsa continuar de lo contrario lo mismo que CALL nn	•	•	•	•	•	•	11 cc 100 ← n → ← n →	3 3	3 5	10 17	Si cc es falsa Si cc es verdadera
RET	PC_L←(SP) PC_H←(SP+1)	•	•	•	•	•	•	11 001 001	1	3	10	
RET cc	Si la condición es falsa continuar, de lo contrario lo mismo que RET	•	•	•	•	•	•	11 cc 000	1 1	1 3	5 11	Si cc es falsa Si cc es verdadera
RETI	Retorno de una interrupción	•	•	•	•	•	•	11 101 101 01 001 101	2	4	14	
RETN	Retorno de una interrupción no enmascarable	•	•	•	•	•	•	11 101 101 01 000 101	2	4	14	
RST p	(SP-1)←PC_H (SP-2)←PC_L PC_H←0 PC_L←P	•	•	•	•	•	•	11 t 111	1	3	11	

cc	Condición
000	NZ no cero
001	Z cero
010	NC no arrastre
011	C arrastre
100	PO paridad impar
101	PE paridad par
110	P signo positivo
111	M signo negativo

t	P
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

Tabla A-11. Grupo input (entrada) y output (salida)

Mnemónico	Operación simbólica	Indicadores C	Z	P/V	S	N	H	Código de operación 76 543 210	Número de bytes	Número de ciclos M	Número de estados T	Comentarios
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011	2	3	10	n a $A_0 \sim A_7$
								← n →				Acc a $A_8 \sim A_{15}$
IN r, (C)	r ← (C)	•	↕	P	↕	0	↕	11 101 101	2	3	11	C a $A_0 \sim A_7$
	Si r = 110 solamente quedarán afectados los indicadores							01 r 000				B to $A_8 \sim A_{15}$
INI	(HL) ← (C)	•	↕ ①	X	X	1	X	11 101 101	2	4	15	C a $A_0 \sim A_7$
	B ← B - 1							10 100 010				B a $A_8 \sim A_{15}$
	HL ← HL + 1											
INIR	(HL) ← (C)	•	1	X	X	1	X	11 101 101	2	5	20	C a $A_0 \sim A_7$
	B ← B - 1							10 110 010		(Si B ≠ 0)		B a $A_8 \sim A_{15}$
	HL ← HL + 1								2	4	15	
	Repetir hasta que B = 0									(Si B = 0)		
IND	(HL) ← (C)	•	↕ ①	X	X	1	X	11 101 101	2	4	15	C a $A_0 \sim A_7$
	B ← B - 1							10 101 010				B a $A_8 \sim A_{15}$
	HL ← HL - 1											
INDR	(HL) ← (C)	•	1	X	X	1	X	11 101 101	2	5	20	C a $A_0 \sim A_7$
	B ← B - 1							10 111 010		(Si B ≠ 0)		B a $A_8 \sim A_{15}$
	HL ← HL -1								2	4	15	
	Repetir hasta que B = 0									(Si B = 0)		
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011	2	3	11	n a $A_0 \sim A_7$
												Acc a $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101	2	3	12	C a $A_0 \sim A_7$
								01 r 001				B a $A_8 \sim A_{15}$
OUTI	(C) ← (HL)	•	↕ ①	X	X	1	X	11 101 101	2	4	15	C a $A_0 \sim A_7$
	B ← B - 1							10 100 011				B a $A_8 \sim A_{15}$
	HL ← HL + 1											
OTIR	(C) ← (HL)	•	1	X	X	1	X	11 101 101	2	5	20	C a $A_0 \sim A_7$
	B ← B - 1							10 110 011		(Si B ≠ 0)		B a $A_8 \sim A_{15}$
	HL ← HL + 1								2	4	15	
	Repetir hasta que B = 0									(Si B = 0)		
OUTD	(C) ← (HL)	•	↕ ①	X	X	1	X	11 101 101	2	4	15	C a $A_0 \sim A_7$
	B ← B - 1							10 101 011				B a $A_8 \sim A_{15}$
	HL ← HL 1											
OTDR	(C) ← (HL)	•	1	X	X	1	X	11 101 101	2	5	20	C a $A_0 \sim A_7$
	B ← B - 1							10 111 011		(Si B ≠ 0)		B a $A_8 \sim A_{15}$
	HL ← HL -1								2	4	15	
	Repetir hasta que B = 0									(Si B = 0)		

Notas: ① Si el resultado de B−1 es cero el indicador Z se coloca a 1, de lo contrario a cero.

Notación de los indicadores: • = indicador no afectado; 0 = indicador colocado a cero; 1 = = indicador colocado a uno; X = indicador desconocido; ↕ = el indicador queda afectado de acuerdo con el resultado de la operación.

Cortesía Zilog, Inc.

APENDICE **B**

Instrucciones de la CPU Z-80 clasificadas por mnemónico

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
00	NOP	218405	LD HL, NN	42	LD B, D
018405	LD BC, NN	228405	LD (NN), HL	43	LD B, E
02	LD (BC), A	23	INC HL	44	LD B, H, NN
03	INC BC	24	INC H	45	LD B, L
04	INC B	25	DEC H	46	LD B, (HL)
05	DEC B	2620	LD H, N	47	LD B, A
0620	LD B, N	27	DAA	48	LD C, B
07	RLCA	282E	JR Z, DIS	49	LD C, C
08	EX AF, AF'	29	ADD HL, HL	4A	LD C, D
09	ADD HL, BC	2A8405	LD (HL), (NN)	4B	LD C, E
0A	LD A,(BC)	2B	DEC HL	4C	LD C, H
0B	DEC BC	2C	INC L	4D	LD C, L
0C	INC C	2D	DEC L	4E	LD C, (HL)
0D	DEC C	2E20	LD L, N	4F	LD C, A
0E20	LD C, N	2F	CPL	50	LD D, B
0F	RRCA	302E	JR NC, DIS	51	LD D, C
102E	DJNZ DIS	318405	LD SP, NN	52	LD D, D
118405	LD DE, NN	328405	LD (NN), A	53	LD D, E
12	LD (DE), A	33	INC SP	54	LD D, H
13	INC DE	34	INC (HL)	55	LD D, L
14	INC D	35	DEC (HL)	56	LD D, (HL)
15	DEC D	3620	LD (HL), N	57	LD D, A
1620	LD D, N	37	SCF	58	LD E, B
17	RLA	382E	JR C, DIS	59	LD E, C
182E	JR DIS	39	ADD HL, SP	5A	LD E, D
19	ADD HL, DE	3A8405	LD A, (NN)	5B	LD E, E
1A	LD A, (DE)	3B	DEC SP	5C	LD E, H
1B	DEC DE	3C	INC A	5D	LD E, L
1C	INC E	3D	DEC A	5E	LD E, (HL)
1D	DEC E	3E20	LD A, N	5F	LD E, A
1E20	LD E, N	3F	CCF	60	LD H, B
1F	RRA	40	LD B, B	61	LD H, C
202E	JR NZ, DIS	41	LD B, C	62	LD H, D

Cortesía Zilog, Inc.

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
63	LD H, E	A5	AND L	E9	JP (HL)
64	LD H, H	A6	AND (HL)	EA8405	JE PE NN
65	LD H, L	A7	AND A	EB	EX DE, HL
66	LD H, (HL)	A8	XOR B	EC8405	CALL PE, NN
67	LD H, A	A9	XOR C	EE20	XOR N
68	LD L, B	AA	XOR D	EF	RST 28H
69	LD L, C	AB	XOR E	F0	RET P
6A	LD L, D	AC	XOR H	F1	POP AF
6B	LD L, E	AD	XOR L	F28405	JP P, NN
6C	LD L, H	AE	XOR (HL)	F3	DI
6D	LD L, L	AF	XOR A	F48405	CALL P, NN
6E	LD L, (HL)	B0	OR B	F5	PUSH AF
6F	LD L, A	B1	OR C	F620	OR N
70	LD (HL), B	B2	OR D	F7	RST 30H
71	LD (HL), C	B3	OR E	F8	RET M
72	LD (HL), D	B4	OR H	F9	LD SP, HL
73	LD (HL), E	B5	OR L	FA8405	JP M, NN
74	LD (HL), H	B6	OR (HL)	FB	EI
75	LD (HL), L	B7	OR A	FC8405	CALL M, NN
76	HALT	B8	CP B	FE20	CP N
77	LD (HL), A	B9	CP C	FF	RST 38H
78	LD A, B	BA	CP D	CB00	RLC B
79	LD A, C	BB	CP E	CB01	RLC C
7A	LD A, D	BC	CP H	CB02	RLC D
7B	LD A, E	BD	CP L	CB03	RLC E
7C	LD A, H	BE	CP (HL)	CB04	RLC H
7D	LD A, L	BF	CP A	CB05	RLC L
7E	LD A, (HL)	C0	RET NZ	CB06	RLC (HL)
7F	LD A, A	C1	POP BC	CB07	RLC A
80	ADD A, B	C28405	JP NZ, NN	CB08	RRC B
81	ADD A, C	C38405	JP NN	CB09	RRC C
82	ADD A, D	C48405	CALL NZ, NN	CB0A	RRC D
83	ADD A, E	C5	PUSH BC	CB0B	RRC E
84	ADD A, H	C620	ADD A, N	CB0C	RRC H
85	ADD A, L	C7	RST O	CB0D	RRC L
86	ADD A, (HL)	C8	RET Z	CB0E	RRC (HL)
87	ADD A, A	C9	RET	CB0F	RRC A
88	ADC A, B	CA8405	JP Z, NN	CB10	RL B
89	ADC A, C	CC8405	CALL Z, NN	CB11	RL C
8A	ADC A, D	CD8405	CALL NN	CB12	RL D
8B	ADC A, E	CE20	ADC A, N	CB13	RL E
8C	ADC A, H	CF	RST 8	CB14	RL H
8D	ADC A, L	D0	RET NC	CB15	RL L
8E	ADC A, (HL)	D1	POP DE	CB16	RL (HL)
8F	ADC A, A	D28405	JP NC, NN	CB17	RL A
90	SUB B	D320	OUT (N), A	CB18	RR B
91	SUB C	D48405	CALL NC, NN	CB19	RR C
92	SUB D	D5	PUSH DE	CB1A	RR D
93	SUB E	D620	SUB N	CB1B	RR E
94	SUB H	D7	RST 10H	CB1C	RR H
95	SUB L	D8	RET C	CB1D	RR L
96	SUB (HL)	D9	EXX	CB1E	RR (HL)
97	SUB A	DA8405	JP C, NN	CB1F	RR A
98	SBC A, B	DB20	IN A, (N)	CB20	SLA B
99	SBC A, C	DC8405	CALL C, N	CB21	SLA C
9A	SBC A, D	DE20	SBC A, N	CB22	SLA D
9B	SBC A, E	DF	RST 18H	CB23	SLA E
9C	SBC A, H	E0	RET PO	CB24	SLA H
9D	SBC A, L	E1	POP HL	CB25	SLA L
9E	SBC A, (HL)	E28405	JP PO, NN	CB26	SLA (HL)
9F	SBC A, A	E3	EX (SP), HL	CB27	SLA A
A0	AND B	E48405	CALL PO, NN	CB28	SRA B
A1	AND C	E5	PUSH HL	CB29	SRA C
A2	AND D	E620	AND N	CB2A	SRA D
A3	AND E	E7	RST 20 H	CB2B	SRA E
A4	AND H	E8	RET PE	CB2C	SRA H

Cortesía Zilog, Inc.

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
CB2D	SRA L	CB77	BIT 6, A	CBB9	RES 7, C
CB2E	SRA (HL)	CB78	BIT 7, B	CBBA	RES 7, D
CB2F	SRA A	CB79	BIT 7, C	CBBB	RES 7, E
CB38	SRL B	CB7A	BIT 7, D	CBBC	RES 7, H
CB39	SRL C	CB7B	BIT 7, E	CBBD	RES 7, L
CB3A	SRL D	CB7C	BIT 7, H	CBBE	RES 7, (HL)
CB3B	SRL E	CB7D	BIT 7, L	CBBF	RES 7, A
CB3C	SRL H	CB7E	BIT 7, (HL)	CBC0	SET 0, B
CB3D	SRL L	CB7F	BIT 7, A	CBC1	SET 0, C
CB3E	SRL (HL)	CB80	RES 0, B	CBC2	SET 0, D
CB3F	SRL A	CB81	RES 0, C	CBC3	SET 0, E
CB40	BIT 0, B	CB82	RES 0, D	CBC4	SET 0, H
CB41	BIT 0, C	CB83	RES 0, E	CBC5	SET 0, L
CB42	BIT 0, D	CB84	RES 0, H	CBC6	SET 0, (HL)
CB43	BIT 0, E	CB85	RES 0, L	CBC7	SET 0, A
CB44	BIT 0, H	CB86	RES 0, (HL)	CBC8	SET 1, B
CB45	BIT 0, L	CB87	RES 0, A	CBC9	SET 1, C
CB46	BIT 0, (HL)	CB88	RES 1, B	CBCA	SET 1, D
CB47	BIT 0, A	CB89	RES 1, C	CBCB	SET 1, E
CB48	BIT 1, B	CB8A	RES 1, D	CBCC	SET 1, H
CB49	BIT 1, C	CB8B	RES 1, E	CBCD	SET 1, L
CB4A	BIT 1, D	CB8C	RES 1, H	CBCE	SET 1, (HL)
CB4B	BIT 1, E	CB8D	RES 1, L	CBCF	SET 1, A
CB4C	BIT 1, H	CB8E	RES 1, (HL)	CBD0	SET 2, B
CB4D	BIT 1, L	CB8F	RES 1, A	CBD1	SET 2, C
CB4E	BIT 1, (HL)	CB90	RES 2, B	CBD2	SET 2, D
CB4F	BIT 1, A	CB91	RES 2, C	CBD3	SET 2, E
CB50	BIT 2, B	CB92	RES 2, D	CBD4	SET 2, H
CB51	BIT 2, C	CB93	RES 2, E	CBD5	SET 2, L
CB52	BIT 2, D	CB94	RES 2, H	CBD6	SET 2, (HL)
CB53	BIT 2, E	CB95	RES 2, L	CBD7	SET 2, A
CB54	BIT 2, H	CB96	RES 2, (HL)	CBD8	SET 3, B
CB55	BIT 2, L	CB97	RES 2, A	CBD9	SET 3,C
CB56	BIT 2, (HL)	CB98	RES 3, B	CBDA	SET 3, D
CB57	BIT 2, A	CB99	RES 3, C	CBDB	SET 3, E
CB58	BIT 3, B	CB9A	RES 3, D	CBDC	SET 3, H
CB59	BIT 3, C	CB9B	RES 3, E	CBDD	SET 3, L
CB5A	BIT 3, D	CB9C	RES 3, H	CBDE	SET 3, (HL)
CB5B	BIT 3, E	CB9D	RES 3, L	CBDF	SET 3, A
CB5C	BIT 3, H	CB9E	RES 3, (HL)	CBE0	SET 4, B
CB5D	BIT 3, L	CB9F	RES 3, A	CBE1	SET 4, C
CB5E	BIT 3, (HL)	CBA0	RES 4, B	CBE2	SET 4, D
CB5F	BIT 3, A	CBA1	RES 4, C	CBE3	SET 4, E
CB60	BIT 4, B	CBA2	RES 4, D	CBE4	SET 4, H
CB61	BIT 4, C	CBA3	RES 4, E	CBE5	SET 4, L
CB62	BIT 4, D	CBA4	RES 4, H	CBE6	SET 4, (HL)
CB63	BIT 4, E	CBA5	RES 4, L	CBE7	SET 4, A
CB64	BIT 4, H	CBA6	RES 4, (HL)	CBE8	SET 5, B
CB65	BIT 4, L	CBA7	RES 4, A	CBE9	SET 5, C
CB66	BIT 4, (HL)	CBA8	RES 5, B	CBEA	SET 5, D
CB67	BIT 4, A	CBA9	RES 5, C	CBEB	SET 5, E
CB68	BIT 5, B	CBAA	RES 5, D	CBEC	SET 5, H
CB69	BIT 5, C	CBAB	RES 5, E	CBED	SET 5, L
CB6A	BIT 5, D	CBAC	RES 5, H	CBEE	SET 5, (HL)
CB6B	BIT 5, E	CBAD	RES 5, L	CBEF	SET 5, A
CB6C	BIT 5, H	CBAE	RES 5, (HL)	CBF0	SET 6, B
CB6D	BIT 5, L	CBAF	RES 5, A	CBF1	SET 6, C
CB6E	BIT 5, (HL)	CBB0	RES 6, B	CBF2	SET 6, D
CB6F	BIT 5, A	CBB1	RES 6, C	CBF3	SET 6, E
CB70	BIT 6, B	CBB2	RES 6, D	CBF4	SET 6, H
CB71	BIT 6, C	CBB3	RES 6, E	CBF5	SET 6, L
CB72	BIT 6, D	CBB4	RES 6, H	CBF6	SET 6, (HL)
CB73	BIT 6, E	CBB5	RES 6, L	CBF7	SET 6, A
CB74	BIT 6, H	CBB6	RES 6, (HL)	CBF8	SET 7, B
CB75	BIT 6, L	CBB7	RES 6, A	CBF9	SET 7, C
CB76	BIT 6, (HL)	CBB8	RES 7, B	CBFA	SET 7, D

Cortesía Zilog, Inc.

CODIGO OBJETO	DECLARACION FUENTE
CBFB	SET 7, E
CBFC	SET 7, H
CBFD	SET 7, L
CBFE	SET 7, (HL)
CBFF	SET 7, A
DD09	ADD IX, BC
DD19	ADD IX, DE
DD218405	LD IX, NN
DD228405	LD (NN), IX
DD23	INC IX
DD29	ADD IX, IX
DD2A8405	LD IX, (NN)
DD2B	DEC IX
DD3405	INC (IX + d)
DD3505	DEC (IX + d)
DD360520	LD (IX + d), N
DD39	ADD IX, SP
DD4605	LD B, (IX + d)
DD4E05	LD C, (IX + d)
DD5605	LD D, (IX + d)
DD5E05	LD E, (IX + d)
DD6605	LD H, (IX + d)
DD6E05	LD L, (IX + d)
DD7005	LD (IX + d), B
DD7105	LD (IX + d), C
DD7205	LD (IX + d), D
DD7305	LD (IX + d), E
DD7405	LD (IX + d), H
DD7505	LD (IX + d), L
DD7705	LD (IX + d), A
DD7E05	LD A, (IX + d)
DD8605	ADD A, (IX + d)
DD8E05	ADC A, (IX + d)
DD9605	SUB (IX + d)
DD9E05	SBC A, (IX + d)
DDA605	AND (IX + d)
DDAE05	XOR (IX + d)
DDB605	OR (IX + d)
DDBE05	CP (IX + d)
DDE1	POP IX
DDE3	EX (SP), IX
DDE5	PUSH IX
DDE9	JP (IX)
DDF9	LD SP, IX
DDCB0506	RLC (IX + d)
DDCB050E	RRC (IX + d)
DDCB0516	RL (IX + d)
DDCB051E	RR (IX + d)
DDCB0526	SLA (IX + d)
DDCB052E	SRA (IX + d)
DDCB053E	SRL (IX + d)
DDCB0546	BIT 0, (IX + d)
DDCB054E	BIT 1, (IX + d)
DDCB0556	BIT 2, (IX + d)
DDCB055E	BIT 3, (IX + d)
DDCB0566	BIT 4, (IX + d)
DDCB056E	BIT 5, (IX + d)
DDCB0576	BIT 6, (IX + d)
DDCB057E	BIT 7, (IX + d)
DDCB0586	RES 0, (IX + d)
DDCB058E	RES 1, (IX + d)
DDCB0596	RES 2, (IX + d)
DDCB059E	RES 3, (IX + d)
DDCB05A6	RES 4, (IX + d)
DDCB05AE	RES 5, (IX + d)
DDCB05B6	RES 6, (IX + d)
DDCB05BE	RES 7, (IX + d)
DDCB05C6	SET 0, (IX + d)
DDCB05CE	SET 1, (IX + d)
DDCB05D6	SET 2, (IX + d)
DDCB05DE	SET 3, (IX + d)
DDCB05E6	SET 4, (IX + d)
DDCB05EE	SET 5, (IX + d)
DDCB05F6	SET 6, (IX + d)
DDCB05FE	SET 7, (IX + d)
ED40	IN B, (C)
ED41	OUT (C), B
ED42	SBC HL, BC
ED438405	LD (NN), BC
ED44	NEG
ED45	RETN
ED46	IM 0
ED47	LD I, A
ED48	IN C, (C)
ED49	OUT (C), C
ED4A	ADC HL, BC
ED4B8405	LD BC, (NN)
ED4D	RETI
ED50	IN D, (C)
ED51	OUT (C), D
ED52	SBC HL, DE
ED538405	LD (NN), DE
ED56	IM 1
ED57	LD A, I
ED58	IN E, (C)
ED59	OUT (C), E
ED5A	ADC HL, DE
ED5B8405	LD DE, (NN)
ED5E	IM 2
ED60	IN H, (C)
ED61	OUT (C), H
ED62	SBC HL, HL
ED67	RRD
ED68	IN L, (C)
ED69	OUT (C), L
ED6A	ADC HL, HL
ED6F	RLD
ED72	SBC HL, SP
ED738405	LD (NN), SP
ED78	IN A, (C)
ED79	OUT (C), A
ED7A	ADC HL, SP
ED7B8405	LD SP, (NN)
EDA0	LDI
EDA1	CPI
EDA2	INI
EDA3	OUTI
EDA8	LDD
EDA9	CPD
EDAA	IND
EDAB	OUTD
EDB0	LDIR
EDB1	CPIR
EDB2	INIR
EDB3	OTIR
EDB8	LDDR
EDB9	CPDR
EDBA	INDR
EDBB	OTDR
FD09	ADD IY, BC
FD19	ADD IY, DE
FD218405	LD IY, NN
FD228405	LD (NN), IY
FD23	INC IY
FD29	ADD IY, IY
FD2A8405	LD IY, (NN)
FD2B	DEC IY
FD3405	INC (IY + d)
FD3505	DEC (IY + d)
FD360520	LD (IY + d), N
FD39	ADD IY, SP
FD4605	LD B, (IY + d)
FD4E05	LD C, (IY + d)
FD5605	LD D, (IY + d)
FD5E05	LD E, (IY + d)
FD6605	LD H, (IY + d)
FD6E05	LD L, (IY + d)
FD7005	LD (IY + d), B
FD7105	LD (IY + d), C
FD7205	LD (IY + d), D
FD7305	LD (IY + d), E
FD7405	LD (IY + d), H
FD7505	LD (IY + d), L
FD7705	LD (IY + d), A
FD7E05	LD A, (IY + d)
FD8605	ADD A, (IY + d)
FD8E05	ADC A, (IY + d)
FD9605	SUB (IY + d)
FD9E05	SBC A, (IY + d)
FDA605	AND (IY + d)
FDAE05	XOR (IY + d)
FDB605	OR (IY + d)
FDBE05	CP (IY + d)
FDE1	POP IY
FDE3	EX (SP), IY
FDE5	PUSH IY
FDE9	JP (IY)
FDF9	LD SP, IY
FDCB0506	RLC (IY + d)
FDCB050E	RRC (IY + d)
FDCB0516	RL (IY + d)
FDCB051E	RR (IY + d)
FDCB0526	SLA (IY + d)
FDCB052E	SRA (IY + d)
FDCB053E	SRL (IY + d)
FDCB0546	BIT 0, (IY + d)
FDCB054E	BIT 1, (IY + d)
FDCB0556	BIT 2, (IY + d)
FDCB055E	BIT 3, (IY + d)
FDCB0566	BIT 4, (IY + d)
FDCB056E	BIT 5, (IY + d)
FDCB0576	BIT 6, (IY + d)
FDCB057E	BIT 7, (IY + d)
FDCB0586	RES 0, (IY + d)
FDCB058E	RES 1, (IY + d)
FDCB0596	RES 2, (IY + d)
FDCB059E	RES 3, (IY + d)
FDCB05A6	RES 4, (IY + d)
FDCB05AE	RES 5, (IY + d)
FDCB05B6	RES 6, (IY + d)
FDCB05BE	RES 7, (IY + d)
FDCB05C6	SET 0, (IY + d)
FDCB05CE	SET 1, (IY + d)
FDCB05D6	SET 2, (IY + d)
FDCB05DE	SET 3, (IY + d)
FDCB05E6	SET 4, (IY + d)
FDCB05EE	SET 5, (IY + d)
FDCB05F6	SET 6, (IY + d)
FDCB05FE	SET 7, (IY + d)

Cortesía Zilog, Inc.

APENDICE C

Instrucciones de la CPU Z-80 clasificadas por código de operación

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
8E	ADC A, (HL)	FD09	ADD IY, BC	CB4D	BIT 1, L
DD8E05	ADC A, (IX + d)	FD19	ADD IY, DE	CB56	BIT 2, (HL)
FD8E05	ADC A, (IY + d)	FD29	ADD IY, IY	DDCB0556	BIT 2, (IX + d)
8F	ADC A, A	FD39	ADD IY, SP	FDCB0556	BIT 2, (IY + d)
88	ADC A, B	A6	AND (HL)	CB57	BIT 2, A
89	ADC A, C	DDA605	AND (IX + d)	CB50	BIT 2, B
8A	ADC A, D	FDA605	AND (IY + d)	CB51	BIT 2, C
8B	ADC A, E	A7	AND A	CB52	BIT 2, D
8C	ADC A, H	A0	AND B	CB53	BIT 2, E
8D	ADC A, L	A1	AND C	CB54	BIT 2, H
CE20	ADC A, N	A2	AND D	CB55	BIT 2, L
ED4A	ADC HL, BC	A3	AND E	CB5E	BIT 3, (HL)
ED5A	ADC HL, DE	A4	AND H	DDCB055E	BIT 3, (IX + d)
ED6A	ADC HL, HL	A5	AND L	FDCB055E	BIT 3, (IY + d)
ED7A	ADC HL, SP	E620	AND N	CB5F	BIT 3, A
86	ADD A, (HL)	CB46	BIT 0, (HL)	CB58	BIT 3, B
DD8605	ADD A, (IX + d)	DDCB0546	BIT 0, (IX + d)	CB59	BIT 3, C
FD8605	ADD A, (IY + d)	FDCB0546	BIT 0, (IY + d)	CB5A	BIT 3, D
87	ADD A, A	CB47	BIT 0, A	CB5B	BIT 3, E
80	ADD A, B	CB40	BIT 0, B	CB5C	BIT 3, H
81	ADD A, C	CB41	BIT 0, C	CB5D	BIT 3, L
82	ADD A, D	CB42	BIT 0, D	CB66	BIT 4, (HL)
83	ADD A, E	CB43	BIT 0, E	DDCB0566	BIT 4, (IX + d)
84	ADD A, H	CB44	BIT 0, H	FDCB0566	BIT 4, (IY + d)
85	ADD A, L	CB45	BIT 0, L	CB67	BIT 4, A
C620	ADD A, N	CB4E	BIT 1, (HL)	CB60	BIT 4, B
09	ADD HL, BC	DDCB054E	BIT 1, (IX + d)	CB61	BIT 4, C
19	ADD HL, DE	FDCB054E	BIT 1, (IY + d)	CB62	BIT 4, D
29	ADD HL, HL	CB4F	BIT 1, A	CB63	BIT 4, E
39	ADD HL, SP	BC48	BIT 1, B	CB64	BIT 4, H
DD09	ADD IX, BC	CB49	BIT 1, C	CB65	BIT 4, L
DD19	ADD IX, DE	CB4A	BIT 1, D	CB6E	BIT 5, (HL)
DD29	ADD IX, IX	CB4B	BIT 1, E	DDCB056E	BIT 5, (IX + d)
DD39	ADD IX, SP	CB4C	BIT 1, H	FDCB056E	BIT 5, (IY + d)

Cortesía Zilog, Inc.

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
CB6F	BIT 5, A	DD2B	DEC IX	71	LD (HL), C
CB68	BIT 5, B	FD2B	DEC IY	72	LD (HL), D
CB69	BIT 5, C	2D	DEC L	73	LD (HL), E
CB6A	BIT 5, D	3B	DEC SP	74	LD (HL), H
CB6B	BIT 5, E	F3	DI	75	LD (HL), L
CB6C	BIT 5, H	102E	DJNZ DIS	3620	LD (HL), N
CB6D	BIT 5, L	FB	EI	DD7705	LD (IX + d), A
CB76	BIT 6, (HL)	E3	EX (SP), HL	DD7005	LD (IX + d), B
DDCB0576	BIT 6, (IX + d)	DDE3	EX (SP), IX	DD7105	LD (IX + d), C
FDCB0576	BIT 6, (IY + d)	FDE3	EX (SP), IY	DD7205	LD (IX + d), D
CB77	BIT 6, A	08	EX AF, AF'	DD7305	LD (IX + d), E
CB70	BIT 6, B	EB	EX DE, HL	DD7405	LD (IX + d), H
CB71	BIT 6, C	D9	EXX	DD7505	LD (IX + d), L
CB72	BIT 6, D	76	HALT	DD360520	LD (IX + d), N
CB73	BIT 6, E	ED46	IM 0	FD7705	LD (IY + d), A
CB74	BIT 6, H	ED56	IM 1	FD7005	LD (IY + d), B
CB75	BIT 6, L	ED5E	IM 2	FD7105	LD (IY + d), C
CB7E	BIT 7, (HL)	ED78	IN A, (C)	FD7205	LD (IY + d), D
DDCB057E	BIT 7, (IX + d)	DB20	IN A, (N)	FD7305	LD (IY + d), E
FDCB057E	BIT 7, (IY + d)	ED40	IN B, (C)	FD7405	LD (IY + d), H
CB7F	BIT 7, A	ED48	IN C, (C)	FD7505	LD (IY + d), L
CB78	BIT 7, B	ED50	IN D, (C)	FD360520	LD (IY + d), N
CB79	BIT 7, C	ED58	IN E, (C)	328405	LD (NN), A
CB7A	BIT 7, D	ED60	IN H, (C)	ED438405	LD (NN), BC
CB7B	BIT 7, E	ED68	IN L, (C)	ED538405	LD (NN), DE
CB7C	BIT 7, H	34	INC (HL)	228405	LD (NN), HL
CB7D	BIT 7, L	DD3405	INC (IX + d)	DD228405	LD (NN), IX
DC8405	CALL C, NN	FD3405	INC (IY + d)	FD228405	LD (NN), IY
FC8405	CALL M, NN	3C	INC A	ED738405	LD (NN), SP
D48405	CALL NC, NN	04	INC B	0A	LD A, (BC)
CD8405	CALL NN	03	INC BC	1A	LD A, (DE)
C48405	CALL NZ, NN	0C	INC C	7E	LD A, (HL)
F48405	CALL P, NN	14	INC D	DD7E05	LD A, (IX + d)
EC8405	CALL PE, NN	13	INC DE	FD7E05	LD A, (IY + d)
E48405	CALL PO, NN	1C	INC E	3A8405	LD A, (NN)
CC8405	CALL Z, NN	24	INC H	7F	LD A, A
3F	CCF	23	INC HL	78	LD A, B
BE	CP (HL)	DD23	INC IX	79	LD A, C
DDBE05	CP (IX + d)	FD23	INC IY	7A	LD A, D
FDBE05	CP (IY + d)	2C	INC L	7B	LD A, E
BF	CP A	33	INC SP	7C	LD A, H
B8	CP B	EDAA	IND	ED57	LD A, I
B9	CP C	EDBA	INDR	7D	LD A, L
BA	CP D	EDA2	INI	3E20	LD A, N
BB	CP E	EDB2	INIR	46	LD B, (HL)
BC	CP H	E9	JP (HL)	DD4605	LD B, (IX + d)
BD	CP L	DDE9	JP (IX)	FD4605	LD B, (IY + d)
FE20	CP N	FDE9	JP (IY)	47	LD B, A
EDA9	CPD	DA8405	JP C, NN	40	LD B, B
EDB9	CPDR	FA8405	JP M, NN	41	LD B, C
EDA1	CPI	D28405	JP NC, NN	42	LD B, D
EDB1	CPIR	C38405	JP NN	43	LD B, E
2F	CPL	C28405	JP NZ, NN	44	LD B, H, NN
27	DAA	F28405	JP P, NN	45	LD B, L
35	DEC (HL)	EA8405	JP PE, NN	0620	LD B, N
DD3505	DEC (IX + d)	E28405	JP PO, NN	ED4B8405	LD BC, (NN)
FD3505	DEC (IY + d)	CA8405	JP Z, NN	018405	LD BC, NN
3D	DEC A	382E	JR C, DIS	4E	LD C, (HL)
05	DEC B	182E	JR DIS	DD4E05	LD C, (IX + d)
0B	DEC BC	302E	JR NC, DIS	FD4E05	LD C, (IY + d)
0D	DEC C	202E	JR NZ, DIS	4F	LD C, A
15	DEC D	282E	JR Z, DIS	48	LD C, B
1B	DEC DE	02	LD (BC), A	49	LD C, C
1D	DEC E	12	LD (DE), A	4A	LD C, D
25	DEC H	77	LD (HL), A	4B	LD C, E
2B	DEC HL	70	LD (HL), B	4C	LD C, H

Cortesía Zilog, Inc.

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
4D	LD C, L	DDB605	OR (IX + d)	CB9F	RES 3, A
0E20	LD C, N	FDB605	OR (IY + d)	CB98	RES 3, B
56	LD D, (HL)	B7	OR A	CB99	RES 3, C
DD5605	LD D, (IX + d)	B0	OR B	CB9A	RES 3, D
FD5605	LD D, (IY + d)	B1	OR C	CB9B	RES 3, E
57	LD D, A	B2	OR D	CB9C	RES 3, H
50	LD D, B	B3	OR E	CB9D	RES 3, L
51	LD D, C	B4	OR H	CBA6	RES 4, (HL)
52	LD D, D	B5	OR L	DDCB05A6	RES 4, (IX + d)
53	LD D, E	F620	OR N	FDCB05A6	RES 4, (IY + d)
54	LD D, H	EDBB	OTDR	CBA7	RES 4, A
55	LD D, L	EDB3	OTIR	CBA0	RES 4, B
1620	LD D, N	ED79	OUT (C), A	CBA1	RES 4, C
ED5B8405	LD DE, (NN)	ED41	OUT (C), B	CBA2	RES 4, D
118405	LD DE, NN	ED49	OUT (C), C	CBA3	RES 4, E
5E	LD E, (HL)	ED51	OUT (C), D	CBA4	RES 4, H
DD5E05	LD E, (IX + d)	ED59	OUT (C), E	CBA5	RES 4, L
FD5E05	LD E, (IY + d)	ED61	OUT (C), H	CBAE	RES 5, (HL)
5F	LD E, A	ED69	OUT (C), L	DDCB05AE	RES 5, (IX + d)
58	LD E, B	D320	OUT (N), A	FDCB05AE	RES 5, (IY + d)
59	LD E, C	EDAB	OUTD	CBAF	RES 5, A
5A	LD E, D	EDA3	OUTI	CBA8	RES 5, B
5B	LD E, E	F1	POP AF	CBA9	RES 5, C
5C	LD E, H	C1	POP BC	CBAA	RES 5, D
5D	LD E, L	D1	POP DE	CBAB	RES 5, E
1E20	LD E, N	E1	POP HL	CBAC	RES 5, H
66	LD H, (HL)	DDE1	POP IX	CBAD	RES 5, L
DD6605	LD H, (IX + d)	FDE1	POP IY	CBB6	RES 6, (HL)
FD6606	LD H, (IY + d)	F5	PUSH AF	DDCB05B6	RES 6, (IX + d)
67	LD H, A	C5	PUSH BC	FDCB05B6	RES 6, (IY + d)
60	LD H, B	D5	PUSH DE	CBB7	RES 6, A
61	LD H, C	E5	PUSH HL	CBB0	RES 6, B
62	LD H, D	DDE5	PUSH IX	CBB1	RES 6, C
63	LD H, E	FDE5	PUSH IY	CBB2	RES 6, D
64	LD H, H	CB86	RES 0, (HL)	CBB3	RES 6, E
65	LD H, L	DDCB0586	RES 0, (IX + d)	CBB4	RES 6, H
2620	LD H, N	FDCB0586	RES 0, (IY + d)	CBB5	RES 6, L
2A8405	LD HL, (NN)	CB87	RES 0, A	CBBE	RES 7, (HL)
218405	LD HL, NN	CB80	RES 0, B	DDCB05BE	RES 7, (IX + d)
ED47	LD I, A	CB81	RES 0, C	FDCB05BE	RES 7, (IY + d)
DD2A8405	LD IX, (NN)	CB82	RES 0, D	CBBF	RES 7, A
DD218405	LD IX, NN	CB83	RES 0, E	CBB8	RES 7, B
FD2A8405	LD IY, (NN)	CB84	RES 0, H	CBB9	RES 7, C
FD218405	LD IY, NN	CB85	RES 0, L	CBBA	RES 7, D
6E	LD L, (HL)	CB8E	RES 1, (HL)	CBBB	RES 7, E
DD6E05	LD L, (IX + d)	DDCB058E	RES 1, (IX + d)	CBBC	RES 7, H
FD6E05	LD L, (IY + d)	FDCB058E	RES 1, (IY + d)	CBBD	RES 7, L
6F	LD L, A	CB8F	RES 1, A	C9	RET
68	LD L, B	CB88	RES 1, B	D8	RET C
69	LD L, C	CB89	RES 1, C	F8	RET M
6A	LD L, D	CB8A	RES 1, D	D0	RET NC
6B	LD L, E	CB8B	RES 1, E	C0	RET NZ
6C	LD L, H	CB8C	RES 1, H	F0	RET P
6D	LD L, L	CB8D	RES 1, L	E8	RET PE
2E20	LD L, N	CB96	RES 2, (HL)	E0	RET PO
ED7B8405	LD SP, (NN)	DDCB0596	RES 2, (IX + d)	C8	RET Z
F9	LD SP, HL	FDCB0596	RES 2, (IY + d)	ED4D	RETI
DDF9	LD SP, IX	CB97	RES 2, A	ED45	RETN
FDF9	LD SP, IY	CB90	RES 2, B	CB16	RL (HL)
318405	LD SP, NN	CB91	RES 2, C	DDCB0516	RL (IX + d)
EDA8	LDD	CB92	RES 2, D	FDCB0516	RL (IY + d)
EDB8	LDDR	CB93	RES 2, E	CB17	RL A
EDA0	LDI	CB94	RES 2, H	CB10	RL B
EDB0	LDIR	CB95	RES 2, L	CB11	RL C
ED44	NEG	CB9E	RES 3, (HL)	CB12	RL D
00	NOP	DDCB059E	RES 3, (IX + d)	CB13	RL E
B6	OR (HL)	FDCB059E	RES 3, (IY + d)		

Cortesía Zilog, Inc.

CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE	CODIGO OBJETO	DECLARACION FUENTE
CB14	RL H	CBC0	SET 0, B	CBFE	SET 7, (HL)
CB15	RL L	CBC1	SET 0, C	DDCB05FE	SET 7, (IX + d)
17	RLA	CBC2	SET 0, D	FDCB05FE	SET 7, (IY + d)
CB06	RLC (HL)	CBC3	SET 0, E	CBFF	SET 7, A
DDCB0506	RLC (IX + d)	CBC4	SET 0, H	CBF8	SET 7, B
FDCB0506	RLC (IY + d)	CBC5	SET 0, L	CBF9	SET 7, C
CB07	RLC A	CBCE	SET 1, (HL)	CBFA	SET 7, D
CB00	RLC B	DDCB05CE	SET 1, (IX + d)	CBFB	SET 7, E
CB01	RLC C	FDCB05CE	SET 1, (IY + d)	CBFC	SET 7, H
CB02	RLC D	CBCF	SET 1, A	CBFD	SET 7, L
CB03	RLC E	CBC8	SET 1, B	CB26	SLA (HL)
CB04	RLC H	CBC9	SET 1, C	DDCB0526	SLA (IX + d)
CB05	RLC L	CBCA	SET 1, D	FDCB0526	SLA (IY + d)
07	RLCA	CBCB	SET 1, E	CB27	SLA A
ED6F	RLD	CBCC	SET 1, H	CB20	SLA B
CB1E	RR (HL)	CBCD	SET 1, L	CB21	SLA C
DDCB051E	RR (IX + d)	CBD6	SET 2, (HL)	CB22	SLA D
FDCB051E	RR (IY + d)	DDCB05D6	SET 2, (IX + d)	CB23	SLA E
CB1F	RR A	FDCB05D6	SET 2, (IY + d)	CB24	SLA H
CB18	RR B	CBD7	SET 2, A	CB25	SLA L
CB19	RR C	CBD0	SET 2, B	CB2E	SRA (HL)
CB1A	RR D	CBD1	SET 2, C	DDCB052E	SRA (IX + d)
CB1B	RR E	CBD2	SET 2, D	FDCB052E	SRA (IY + d)
CB1C	RR H	CBD3	SET 2, E	CB2F	SRA A
CB1D	RR L	CBD4	SET 2, H	CB28	SRA B
1F	RRA	CBD5	SET 2, L	CB29	SRA C
CB0E	RRC (HL)	CBD8	SET 3, B	CB2A	SRA D
DDCB050E	RRC (IX + d)	CBDE	SET 3, (HL)	CB2B	SRA E
FDCB050E	RRC (IY + d)	DDCB05DE	SET 3, (IX + d)	CB2C	SRA H
CB0F	RRC A	FDCB05DE	SET 3, (IY + d)	CB2D	SRA L
CB08	RRC B	CBDF	SET 3, A	CB3E	SRL (HL)
CB09	RRC C	CBD9	SET 3, C	DDCB053E	SRL (IX + d)
CB0A	RRC D	CBDA	SET 3, D	FDCB053E	SRL (IY + d)
CB0B	RRC E	CBDB	SET 3, E	CB3F	SRL A
CB0C	RRC H	CBDC	SET 3, H	CB38	SRL B
CB0D	RRC L	CBDD	SET 3, L	CB39	SRL C
0F	RRCA	CBE6	SET 4, (HL)	CB3A	SRL D
ED67	RRD	DDCB05E6	SET 4, (IX + d)	CB3B	SRL E
C7	RST 0	FDCB05E6	SET 4, (IY + d)	CB3C	SRL H
D7	RST 10H	CBE7	SET 4, A	CB3D	SRL L
DF	RST 18H	CBE0	SET 4, B	96	SUB (HL)
E7	RST 20H	CBE1	SET 4, C	DD9605	SUB (IX + d)
EF	RST 28H	CBE2	SET 4, D	FD9605	SUB (IY + d)
F7	RST 30H	CBE3	SET 4, E	97	SUB A
FF	RST 38H	CBE4	SET 4, H	90	SUB B
CF	RST 8	CBE5	SET 4, L	91	SUB C
9E	SBC A, (HL)	CBEE	SET 5, (HL)	92	SUB D
DD9E05	SBC A, (IX + d)	DDCB05EE	SET 5, (IX + d)	93	SUB E
FD9E05	SBC A, (IY + d)	FDCB05EE	SET 5, (IY + d)	94	SUB H
9F	SBC A, A	CBEF	SET 5, A	95	SUB L
98	SBC A, B	CBE8	SET 5, B	D620	SUB N
99	SBC A, C	CBE9	SET 5, C	AE	XOR (HL)
9A	SBC A, D	CBEA	SET 5, D	DDAE05	XOR (IX + d)
9B	SBC A, E	CBEB	SET 5, E	FDAE05	XOR (IY + d)
9C	SBC A, H	CBEC	SET 5, H	AF	XOR A
9D	SBC A, L	CBED	SET 5, L	A8	XOR B
DE20	SBC A, N	CBF6	SET 6, (HL)	A9	XOR C
ED42	SBC HL, BC	DDCB05F6	SET 6, (IX + d)	AA	XOR D
ED52	SBC HL, DE	FDCB05F6	SET 6, (IY + d)	AB	XOR E
ED62	SBC HL, HL	CBF7	SET 6, A	AC	XOR H
ED72	SBC HL, SP	CBF0	SET 6, B	AD	XOR L
37	SCF	CBF1	SET 6, C	EE20	XOR N
CBC6	SET 0, (HL)	CBF2	SET 6, D		
DDCB05C5	SET 0, (IX + d)	CBF3	SET 6, E		
FDCB05C6	SET 0, (IY + d)	CBF4	SET 6, H		
CBC7	SET 0, A	CBF5	SET 6, L		

Cortesía Zilog, Inc.

APENDICE **D**

Cálculo de los tiempos de ejecución

La siguiente información se refiere a la pregunta de como calcular los tiempos de ejecución para secuencias de instrucciones del Z-80. El tiempo de ejecución es normalmente una cualidad importante de un programa y que debe ser considerado cuando se seleccionan métodos alternativos de implementación.

Considere la siguiente secuencia de instrucciones:

```
LD   A, 36H
LD   B, 49H
OR   B
AND  99H
RL   A
```

¿Cuánto tiempo tardará su Nanocomputador en ejecutar estas instrucciones? Para determinar la respuesta a esta pregunta usted debe conocer la velocidad del reloj externo del Nanocomputador. Esta es de 2,5 MHz o 2,5 megahertzios o 2.500.000 ciclos por segundo. Esto es, cada ciclo dura

$$\frac{1}{2{,}5 \times 10^6} \text{ segundo} = 0{,}0000004 \text{ segundo}$$

Puesto que 1 segundo = 10^3 milisegundos (ms) = 10^6 microsegundos (μs) = 10^9 nanosegundos (ns), el tiempo de ciclo de su Nanocomputador es de 0,0004 ms o 0,4 μs o 400 ns. Algunas CPU del Z-80 seleccionadas especialmente pueden funcionar a 4 MHz, o a un tiempo de ciclo de 250 ns. Las tablas en el apéndice A dan el número de estados T, o ciclos externos de reloj, necesarios para ejecutar cada

instrucción del Z-80. Así utilizando estas tablas podemos hacer los siguientes cálculos.

Instrucción	Núm. estados T	Núm. de veces ejecutada	Tiempo (μs) total de ejecución
LD A,36H	7	1	7 estados T = 2,8
LD B,49H	7	1	2,8
OR B	4	1	1,6
AND 99H	7	1	2,8
RL A	4	1	1,6

Así el tiempo de ejecución para la secuencia de instrucciones es de 11,6 μs.

Para el ejemplo anterior, el número de tiempos que se ejecuta cada instrucción es siempre 1. Observemos el bucle de retardo de nuestro próximo ejemplo.

```
      LD   A,06H
      LD   B,08H
LOOP: INC  A
      DEC  B
      JP   NZ,LOOP
```

El cálculo de tiempo para esta secuencia de instrucciones es el siguiente:

Instrucción	Núm. estados T	Núm. de veces ejecutada	Tiempo (μs) total de ejecución
LD A,06H	7	1	2,8
LD B,08H	7	1	2,8
INC A	4	9	14,4
DEC B	4	9	14,4
JP NZ,LOOP	12 (condición alcanzada)	1	4,8
	7 (condición no alcanzada)	8	22,4
		Total:	61,6 μs

Si el anterior bucle de retardo se hubiera diseñado como una subrutina, el retardo provocado por la rutina también habría incluido el tiempo necesario para ejecutar la llamada inicial CALL y la instrucción final de retorno RET:

CALL	17	1	6,8
RET	10	1	4,0
		Nuevo total:	72,4 μs

Vamos ahora a examinar un ejemplo final que utiliza la instrucción LDIR:

```
LD   HL,0100H
LD   DE,0200H
LD   BC,0010H
LDIR
```

Instrucción	Núm. estados T	Núm. de veces ejecutada	Tiempo (μs) total de ejecución
LD HL,0100H	10	1	4,0
LD DE,0200H	10	1	4,0
LD BC,0010H	10	1	4,0
LDIR	21 (Si BC ≠ 0)	15	126,0
	16 (Si BC = 0)	1	6,4
		Total:	144,4 μs

APENDICE E

Precauciones mientras se manipulan dispositivos MOS

Los dispositivos MOS son extremadamente sensibles y se pueden estropear debido a:

- Electricidad estática e
- Inserción incorrecta dentro del zócalo de la tarjeta del Nanocomputador.

Se deben observar las siguientes precauciones para manipular los dispositivos MOS:

1. Asegurarse de que usted está descargado de electricidad estática antes de tocar el dispositivo. Esto se puede lograr tocando con las manos a un material conductor.
2. Evitar tocar las patillas.
3. Evitar que los pins entren en contacto con cualquier material susceptible de almacenar una carga estática, por ejemplo una carpeta de nylon.
4. Si es necesario transportar un dispositivo MOS fuera de su lugar normal de funcionamiento, el dispositivo se debe montar en una esponja conductora para prevenir con eficacia que las patillas puedan quedar sujetas a diferentes potenciales estáticos.
5. Asegúrese de que las partes de recambio están montadas correctamente, por ejemplo la patilla 1 orientada correctamente.

APENDICE F

Tabla de símbolos maestra

Etiqueta	Dirección
BAUDRT	0FAE
CONTST	FB43
INMODE	0FAB

APENDICE **G**

Referencias

1. *The Compact Edition of the Oxford English Dictionary,* Oxford Univ. Press, 1971.
2. Rudolf F. Graf, *Modern Dictionary of Electronics,* Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1977.
3. James Martin, *Telecommunications and the Computer,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.
4. Abraham Marcus y John D. Lenk, *Computers for Technicians,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
5. Microdata Corporation, *Microprogramming Handbook,* Santa Ana, California, 1971.
6. J. Blukis y M. Baker, *Practical Digital Electronics,* Hewlett-Packard Company, Santa Clara, California, 1974.
7. Donald E. Lancaster, *TTL Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1974.
8. H. V. Malmstadt, C. G. Enke, y S. R. Crouch, *Instrumentation for Scientists Series, Module 3. Digital and Analog Data Conversions,* W. A. Benjamin, Inc., Menlo Park, California, 1973-4.
9. H. V. Malmstadt y C. G. Enke, *Digital Electronics for Scientists,* W. A. Benjamin, Inc., New York, 1969.

10. J. D. Lenk, *Handbook of Logic Circuits,* Reston Publishing Company, Inc., Reston, Virginia, 1972.

11. A. James Diefenderfer, *Principles of Electronic Instrumentation,* W. B. Saunders Company, Philadelphia, Pennsylvania, 1972.

12. P. R. Rony y D. G. Larsen, *Logic & Memory Experiments Using TTL Integrated Circuits, Book 2,* Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1979.

13. Robert L. Morris y John R. Miller, Editors, *Designing with TTL Integrated Circuits,* McGraw-Hill Book Company, New York, 1971.

14. Charles J. Sippl, *Microcomputer Dictionary and Guide,* Matrix Publishers, Inc., Champaign, Illinois, 1976.

15. Donald Eadie, *Introduction to the Basic Computer,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

16. Texas Instruments Incorporated, *Microprocessor Handbook,* Dallas, Texas, 1975.

Más acerca del Sistema de Entrenamiento NANOCOMPUTADOR

El Sistema de Entrenamiento Nanocomputador diseñado por SGS-ATES es uno de los pocos sistemas concebidos para educación, para la programación del Z-80 y también para el diseño de circuitos del microcomputador Z-80.

El sistema de entrenamiento es modular y ampliable a una familia completa de microcomputador. Existen tres sistemas básicos:

Hardware

NBZ80: Una única y potente tarjeta, que incluye el "Nanocomputador" y la unidad de entrada/display, con un sistema operativo de 2K incluido en la tarjeta, que se puede ampliar mediante un kit a una tarjeta completa de microcomputador SGS-ATES CLZ80 utilizando ensamblador o BASIC.

NEZ80: Una tarjeta para experimentar que está conectada mediante un interface al bus de señales del NBZ80 y tiene una amplia zona en la tarjeta para montar experimentos con circuitos digitales e interfaces para el microcomputador sin necesidad de efectuar soldaduras.

NPZ80: Una atractiva caja metálica que contiene una fuente de alimentación (±5, ±12 V).

Kits de cables

Además, existe un kit de cables (K1Z80) y un kit de componentes (K2Z80) para su utilización en los experimentos descritos en los libros de entrenamiento.

Libros de entrenamiento

Volumen 1 contiene los conceptos básicos del microprocesador, lenguaje máquina, además de muchos experimentos de programación para ilustrar los conceptos y reforzar el proceso de aprendizaje.

Volumen 2 es una introducción a los circuitos integrados digitales de la familia T14LSxx (Schottky TTL de baja potencia), su funcionamiento y aplicaciones además de muchos experimentos para ilustrar los conceptos y reforzar el proceso de aprendizaje.

Volumen 3 es un estudio completo del interface de Memoria e I/O al microprocesador Z-80 y cubre la PIO Z-80 y los chips CTC además de muchos experimentos extensivos (utilizando cerca de 2K bytes de software en conjunto) para ilustrar los conceptos y reforzar el proceso de aprendizaje.

Los libros de entrenamiento y el hardware son utilizados conjuntamente de la siguiente forma:

	NBZ80	NEZ80	NPZ80	K1Z80	K2Z80
Vol. 1	X		X		
Vol. 2		X	X	X	X
Vol. 3	X	X	X	X	X

NOTA: De los citados volúmenes 1, 2 y 3 existe solamente traducción al castellano el presente que corresponde al número 1.

marcombo
PROGRAMACION DEL MICROPROCESADOR Z-80
•E.A. NICHOLS
•J.C. NICHOLS
•P.R. RONY